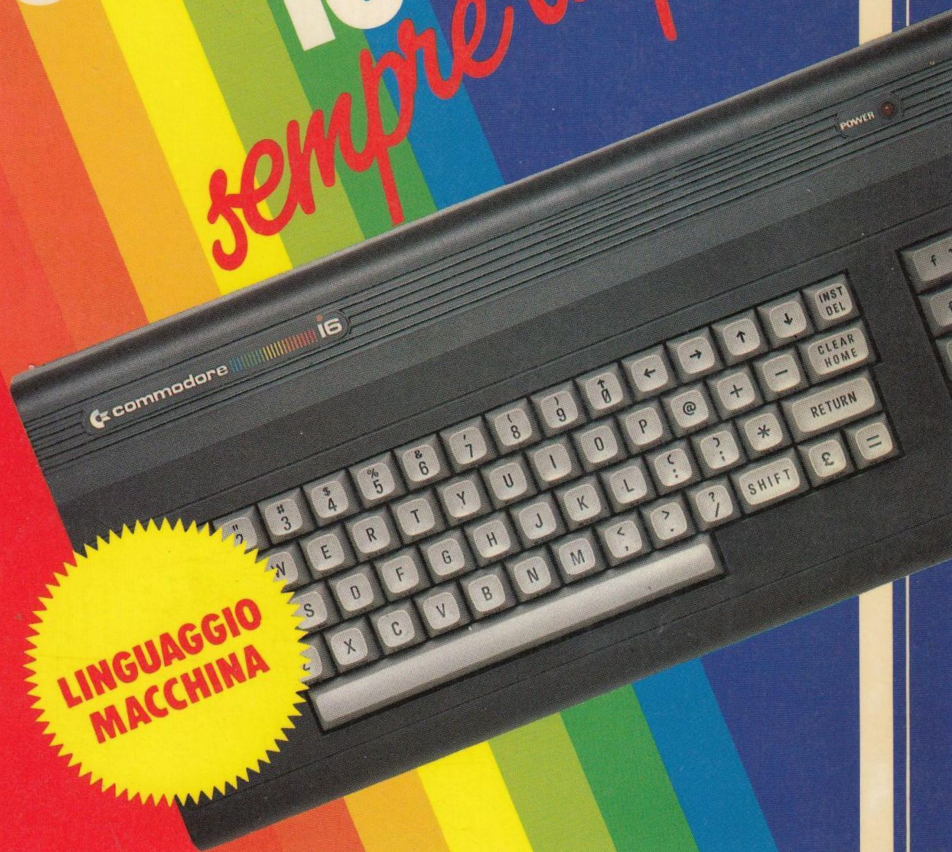
Rita Bonelli
Luciano Pazzucconi
Fabio Racchi

COMMODORE 16 *sempre di più*

LINGUAGGIO MACCHINA

COMMODORE 16 *sempre di più*

Rita Bonelli
Luciano Pazzucconi
Fabio Racchi

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

PREFAZIONE

Questo libro fa seguito al nostro precedente volume "COMMODORE 16 per te". Noi presupponiamo nel lettore una conoscenza elementare del BASIC 3.5, di cui facciamo largo uso nei programmi esempio.

In questo volume approfondiamo la conoscenza delle periferiche collegabili al COMMODORE 16, trattando ampiamente il software per la gestione delle medesime.
Altri argomenti centrali del libro sono l'architettura del sistema calcolatore e la sua programmazione in linguaggio macchina e in ASSEMBLER. Viene trattata con mano leggera, ma con chiarezza, la struttura hardware del calcolatore, mettendo in grado i poco esperti di comprendere meglio le caratteristiche dello strumento. Lo studio del linguaggio macchina consente di sfruttare al meglio le possibilita' del calcolatore, ma anche di comprendere come, per molte operazioni, sia piu' comodo servirsi del linguaggio BASIC.
Diamo ampio spazio all'argomento della grafica, al trattamento dei file su disco, e all'utilizzo della stampante.

Il libro contiene molti programmi commentati, sia in BASIC che in ASSEMBLER, che risultano molto utili, dopo attenta lettura, per imparare a programmare. Questi, inoltre, possono essere usati dal lettore, sia nella versione presentata, che apportando alcune modifiche, o adoperando le routine piu' significative, per il proprio lavoro.

Nella cassetta allegata sono riportati tutti i programmi abbastanza lunghi, quelli cioe' per i quali vale la pena di risparmiare il lavoro di trascrizione.

Gli autori

Gli autori ringraziano James Bachmann, Amministratore Delegato, e Sergio Messa, Direttore Generale, della Commodore Italiana s.p.a., per aver messo a disposizione le apparecchiature e la documentazione necessarie alla realizzazione dell'opera.

Nel testo per esigenze tipografiche la FRECCIA VERSO L'ALTO, che ha il significato di elevamento a potenza, e' stampata col carattere ^.
I numeri esadecimali compaiono con il suffisso "H" o con il prefisso "$", nei punti dove la mancanza dei medesimi potrebbe generare confusione.

SOMMARIO

INDICE FIGURE E TABELLE

CAPITOLO 1

COMUNICAZIONE CON LE PERIFERICHE

1.1 COME IL CALCOLATORE COMUNICA CON LE PERIFERICHE

Le periferiche sono dei dispositivi per l'ingresso e l'uscita dei dati, tramite i quali comunichiamo con il calcolatore. Ogni periferica utilizza un particolare tipo di supporto fisico per la comunicazione.
Alcuni di questi dispositivi di I/O (Input/Output) sono tali da consentire una comunicazione diretta con l'utente; nel caso del COMMODORE 16 abbiamo:
- .la tastiera, che usa il supporto tasti,
- .il televisore (o il monitor), che usa il supporto schermo video,
- .la stampante, che usa il supporto carta,
- .i joystick, che usano come supporto leve o bottoni o rotori.

Altri non consentono una comunicazione diretta, in quanto utilizzano supporti fisici di tipo magnetico; per il COMMODORE 16 abbiamo:
- .il registratore a cassetta,
- .l'unita' a floppy disk.

I dispositivi di I/O sono apparecchiature distinte dal calcolatore, anche se esso, come il COMMODORE 16, si trova inscatolato in una di esse, la tastiera. Per consentire la comunicazione tra il calcolatore e ogni dispositivo di I/O e' necessaria un'interfaccia di collegamento. Le interfacce di I/O sono, in generale, dei dispositivi programmabili, cioe' contengono al loro interno dei registri, ai quali si puo' accedere mediante indirizzi, come se fossero byte di memoria, programmando il funzionamento dell'interfaccia stessa. Dal punto di vista fisico le interfacce di I/O

possono gia' essere contenute all'interno del calcolatore o delle periferiche. Per il COMMODORE 16 le interfacce per le periferiche standard COMMODORE sono gia' contenute nel calcolatore.

Ogni periferica di I/O e' individuata mediante un numero intero, quello che, nella descrizione del BASIC nel primo volume, abbiamo chiamato "unita'". Questo numero nella letteratura viene spesso chiamato "dn", da Device Number (numero apparecchiatura). Per alcune periferiche il numero "dn" puo' essere modificato agendo su alcuni switch, interni o esterni; per altre esso e' fisso. Per il COMMODORE 16 abbiamo:

Periferica	"dn"	Tipo operazione
Tastiera	0	Input
Registratore	1	Input/Output
(RS232 per altri modelli)	2	(non usata)
Video	3	Input/Output
Stampante	4/5	Output
Plotter	6	Output
Disco	8/11	Input/Output

Per i joystick non esiste un "dn", la comunicazione risulta sempre aperta.

Il numero "dn" deve essere usato in alcune istruzioni del BASIC.

Per le periferiche collegate tramite l'interfaccia seriale, stampante e disco, deve essere definito un numero che identifica il modo della comunicazione. Questo numero e' stato chiamato "sa" nella descrizione del BASIC; esso puo' variare da 0 a 15. Il numero "sa" ha diverso significato a seconda delle periferiche; nei Capitoli 2 e 3 trattiamo diffusamente stampante e disco.

L'insieme dei dati che vengono scambiati con una periferica si chiama "stream" o "flusso". Esso si trasforma in Output in una registrazione sul supporto fisico che viene chiamata "file". In Input il file, gia' registrato sul supporto fisico, alimenta il flusso di dati verso il calcolatore. Ogni file e' contraddistinto da un numero, chiamato "lfn", da Logical Fi-

le Number (numero logico file). Il "lfn" puo' variare da 0 a 255; per alcune periferiche una parte dell'intervallo di variabilita' del "lfn", o alcuni valori, possono avere un significato particolare.
Molte istruzioni BASIC usano il "lfn", che viene definito con l'istruzione OPEN.
Un file e' composto da record, cioe' da singoli gruppi di registrazioni che ha senso considerare insieme. Si chiamano campi le singole registrazioni che compongono un record. Dobbiamo fare una distinzione tra:
 .record logici,
 .record fisici.
Le dimensioni dei record, logici e fisici, e dei campi si misurano in numero di caratteri o in byte.
Il RECORD LOGICO e' composto da tutti i campi che interessano l'argomento a cui il record si riferisce; in conseguenza puo' essere lungo quanto e' necessario (almeno in linea teorica).
Il RECORD FISICO dipende dal tipo di supporto di registrazione usato; per esempio su un tipo di carta e con una stampante di un certo tipo non si possono scrivere piu' di 80 caratteri per riga. In questo caso il record fisico, riga di stampa, e' di 80 caratteri. Possiamo usare quella stampante con quella carta per scrivere i dati di un record logico di 300 caratteri; il record logico usera' piu' record fisici. Analogamente una linea del video contiene 40 caratteri, ma noi possiamo scrivere sul video un record logico lungo che sta su piu' righe.
In generale esiste un limite nelle dimensioni dei record fisici, ed esso dipende anche dal supporto di registrazione.
Altre caratteristiche che vengono in generale precisate per i file sono le seguenti:
 .record logici di lunghezza fissa o variabile,
 .numero dei campi, componenti il record, fisso o variabile,
 .lunghezza dei singoli campi fissa o variabile,
 .tipo del file: sequenziale, diretto, relativo, con indice sequenziale,
 .metodo di accesso ai record del file.

Un file e' di tipo SEQUENZIALE quando viene creato scrivendo i record uno dopo l'altro, partendo dal primo. Esso e' di tipo DIRETTO quando si puo' scrivere un record logico conoscendo la sua posizione fisica sul supporto di registrazione, indipendentemente dalla scrittura del record precedente. Un file RELATIVO vie-

ne scritto record per record (anche non in sequenza) fornendo il numero d'ordine del record nel file (primo, terzo,...). Un file CON INDICE SEQUENZIALE viene scritto un record dopo l'altro, creando contemporaneamente un indice sequenziale (file separato dal file principale) che consente l'accesso diretto mediante una chiave al record del file principale; la chiave e' il valore di un determinato campo.
Il metodo di accesso ai record di un file dipende dal tipo di file, dal supporto di registrazione usato e dal software disponibile.
La gestione software delle periferiche puo' avvenire solo per mezzo delle apposite routine del SISTEMA OPERATIVO del calcolatore, oppure puo' dipendere anche da routine memorizzate nelle ROM delle periferiche. Nel nostro caso la stampante e l'unita' disco contengono del software registrato in ROM.

Riepiloghiamo le istruzioni del BASIC che riguardano la gestione delle periferiche come file:

OPEN lfn[,dn[,sa[,"nomef,tipo,modo"]]]
per stabilire la comunicazione

CLOSE lfn
per chiudere la comunicazione

CMD lfn[,lista]
per trasferire a una periferica diversa l'uscita video

GET# lfn,lista variabili
per leggere dati carattere per carattere

INPUT# lfn,lista variabili
per leggere dati variabile per variabile

PRINT# lfn,lista
per scrivere dati

Il sistema raccoglie in una tabella le informazioni relative ai file aperti; tale tabella puo' contenere 10 elementi.

TABELLA GESTIONE FILE			
Num.elem.	lfn	dn	sa
. 1)	1289	1299	1309
. 2)	1290	1300	1310
. 3)	1291	1301	1311
. 4)	1292	1302	1312
. 5)	1293	1303	1313
. 6)	1294	1304	1314
. 7)	1295	1305	1315
. 8)	1296	1306	1316
. 9)	1297	1307	1317
.10)	1298	1308	1318

La tabella dei file aperti viene utilizzata in modo che le informazioni sui file aperti occupino le prime posizioni. Quando un file viene chiuso, se e' l'ultimo nella tabella, essa resta come e' e viene aggiornato solo il puntatore (byte 151) all'ultima posizione occupata, se non e' l'ultimo, la sua posizione viene occupata dall'ultimo file e viene aggiornato il puntatore.

Il programma ES1.4 che segue mostra i contenuti di questa tabella in diverse situazioni, e i contenuti dei byte che danno informazioni circa il file corrente e il numero dei file aperti. Questi ultimi sono:

byte 171: lunghezza nome file corrente
byte 172: lfn file corrente
byte 173: sa file corrente
byte 174: dn file corrente
byte 175/176: puntatore al nome del file
byte 151: numero file aperti e contemporaneamente puntatore all'ultima posizione occupata nella tabella.

```
1 REM ES1.4
2 OPEN4,4:CMD4
3 GOSUB100
10 OPEN0,0
15 OPEN1,3
20 OPEN2,0
21 PRINT#4:CLOSE4
23 OPEN5,1,1,"PIPPO"
24 OPEN4,4:CMD4
25 GOSUB100
30 PRINT#4:CLOSE4
40 CLOSE0:CLOSE1:CLOSE2:CLOSE5
43 OPEN4,4:CMD4
45 GOSUB100
47 PRINT#4:CLOSE4
50 STOP
100 REM STATO FILE
101 PRINT"NUMERO FILE APERTI: ";PEEK(151)
103 PRINT"FILE CORRENTE"
104 PRINT"LFN","DN","SA"
105 PRINTPEEK(172),PEEK(174),PEEK(173)
110 PRINT"PUNTATORE NOME: ";256*PEEK(176)+PEEK(175)
113 PRINT"LUNGHEZZA NOME: ";PEEK(171)
115 PRINT:PRINT"TABELLA FILE"
117 PRINT"LFN","DN","SA"
120 FORK=1289TO1298
125 PRINTPEEK(K),PEEK(K+10),PEEK(K+20)
130 NEXTK
135 RETURN
```

COMMENTO A ES1.4

.2/3: viene aperta la stampante e stampata la situazione.

.10/25: vengono aperti i file con lfn 0, 1 e 2, poi viene chiusa la stampante e aperto un file su cassetta con lfn=5. Viene riaperta la stampante e stampata la situazione.

.30/45: vengono chiusi tutti i file aperti, poi riaperta la stampante e stampata la situazione.

.47/50: viene chiusa la stampante e il programma termina.

.100/135: sottoprogramma che stampa il valore dei puntatori e la tabella.

Osservando i risultati puoi verificare come viene gestita la tabella.

```
NUMERO FILE APERTI:   1
FILE CORRENTE
LFN            DN             SA
 4              4                255
PUNTATORE NOME:   250
LUNGHEZZA NOME:   0

TABELLA FILE
LFN            DN             SA
 4              4                255
 0              0                0
 0              0                0
 0              0                0
 0              0                0
 0              0                0
 0              0                0
 0              0                0
 0              0                0
 0              0                0
```

```
NUMERO FILE APERTI:   5
FILE CORRENTE
LFN          DN          SA
 4            4            255
PUNTATORE NOME:   250
LUNGHEZZA NOME:   0

TABELLA FILE
LFN          DN          SA
 2            0            96
 0            0            96
 1            3            255
 5            1            97
 4            4            255
 0            0            0
 0            0            0
 0            0            0
 0            0            0
 0            0            0
```

```
NUMERO FILE APERTI:  1
FILE CORRENTE
LFN          DN          SA
 4            4           255
PUNTATORE NOME:   250
LUNGHEZZA NOME:   0

TABELLA FILE
LFN          DN          SA
 4            4           255
 5            1           97
 1            3           255
 5            1           97
 4            4           255
 0            0           0
 0            0           0
 0            0           0
 0            0           0
 0            0           0
```

1.2 LA TASTIERA VISTA COME FILE

Il BASIC fornisce le istruzioni per la tastiera, che e' considerata la periferica di Input principale. Il byte 152 contiene il numero di default della periferica di Input; esso all'accensione e' zero, "dn" della tastiera. I comandi INPUT, GET e GETKEY attendono Input dalla tastiera. Essa e' una periferica un po' particolare, infatti pur essendo di Input, le routine del SISTEMA OPERATIVO relative al suo uso forniscono per alcune istruzioni (INPUT) anche un Output sul video, che, ovviamente, potrebbe essere evitato, usando routine diverse.
Vediamo ora, con alcuni esempi, come si puo' usare la tastiera come una periferica gestita come file.

Nel programma ES1.1 mostriamo un primo esempio.

```
1 REM ES1.1
10 OPEN 1,0
15 PRINT"SCRIVI NUMERO: ";:INPUT#1,N
20 PRINT "▶▶▶ HAI SCRITTO: ";N
25 CLOSE1
```

COMMENTO A ES1.1

.10: apriamo la dn=0 come file con lfn=1, trascurando gli altri parametri della OPEN, che non risultano necessari.

.15: con la PRINT scriviamo il messaggio di richiesta di un numero e terminiamo con ";" per non andare a capo. Con la INPUT#1, leggiamo dal file logico 1 un numero, cioe' dalla tastiera. Quando esegui il programma non vedi il "?" di richiesta dati. Termini il numero come al solito con RETURN, vedi scomparire il cursore, ma esso non va a nuova linea.

.20: stampiamo un messaggio e il numero N. Il messaggio tra virgolette inizia con due caratteri "cursore a destra" perche' altrimenti esso verrebbe scritto sopra l'ultima cifra del numero; ne abbiamo messi due per creare anche uno spazio. Possiamo concludere che il RETURN di chiusura dato lascia in questo caso il cursore sull'ultimo carattere scritto.
Se rispondi con un numero di parecchie cifre, il messaggio successivo all'INPUT andra' in parte a nuova linea.

Nel programma ES1.2 riportiamo un altro esempio.

```
1 REM ES1.2
10 OPEN 170,0
15 PRINT"SCRIVI NUMERO: ";:INPUT#170,N
20 PRINT:PRINT "HAI SCRITTO: ";N
25 CLOSE170
```

La differenza di ES1.2 rispetto al programma precedente e' che abbiamo usato lfn=170 e che alla linea 20, andiamo a capo con la prima PRINT e, poi, scriviamo il messaggio; abbiamo potuto eliminare i caratteri di "cursore a destra".

Nel programma ES1.3 riportiamo un terzo esempio.

```
1 REM ES1.3
10 OPEN 0,0
15 PRINT"SCRIVI NUMERO: ";:INPUT#0,N
20 PRINT "HAI SCRITTO: ";N
25 CLOSE0
```

In questo caso abbiamo usato lfn=0, questo produce il solito effetto di INPUT dalla tastiera, cioe' compare il punto interrogativo di richiesta dati e il RETURN di chiusura dato manda a nuova linea. L'istruzione INPUT normale corrisponde ad aver usato una OPEN 0,0 seguita da una INPUT#0.

1.3 IL VIDEO VISTO COME FILE

L'istruzione PRINT del BASIC manda i dati sul video, che e' considerato la periferica di Output principale, a partire dalla posizione del cursore; la posizione di stampa puo' essere modificata usando i caratteri di controllo del cursore e le funzioni TAB e SPC.
Il byte 153 contiene il numero di default della periferica di Output; esso all'accensione e' 3, "dn" del video.
Vediamo ora, con alcuni esempi, come si puo' usare il video, gestendolo come un file, sia in Input che in Output.
Nel programma ES1.5 mostriamo un primo esempio.

```
1 REM ES1.5
10 OPEN3,3
15 PRINT#3,"■";"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
16 PRINT#3,1234
17 A$=""
20 PRINT#3,"■";
23 FORK=1TO26:GET#3,B$
25 A$=A$+B$
30 NEXTK:PRINT#3,"■■";
31 N$="":GET#3,N1$
33 GET#3,N1$:IFN1$=CHR$(32)THEN35
34 N$=N$+N1$:GOTO33
35 PRINT#3,"■■■■■■■■■■■■■"A$
37 PRINT#3,VAL(N$)
40 CLOSE3:STOP
```

COMMENTO A ES1.5

.10: apre il video, dn=3, con lfn=3.
.15: stampa sulla prima linea del video pulito, dopo aver usato il carattere di controllo SHIFT-CLEAR/HOME, le 26 lettere dell'alfabeto, con PRINT#3.

.16: stampa con PRINT#3 sulla seconda linea del video il numero 1234. Nota che dopo la stampa il cursore va a capo, ma sul video non viene registrato il carattere CHR$(13). Il video e' stato usato come file di Output.

.17: pone A$ a stringa nulla.

.20: stampa con PRINT#3 il carattere di controllo CLR/HOME per portare il cursore nella prima posizione del video.

.23/30: legge con GET#3, quindi usa il video come file di Input, quello che sta sulla prima linea, carattere per carattere e somma i caratteri letti in A$; questo per 26 volte. Poi con PRINT#3 stampa i caratteri di controllo CLR/HOME e FRECCIA GIU', per posizionarsi all'inizio della seconda riga del video.

.31: pone N$ a stringa nulla e legge il primo carattere, che sappiamo essere uno spazio, a vuoto. Se vogliamo leggere un numero con il segno meno, questo primo carattere va posto nella stringa N$.

.33: legge in N1$ un carattere; se esso e' uno spazio, cioe' il numero e' terminato, prosegue dalla linea 35.

.34: somma il carattere letto nella stringa N$ e torna alla linea 33.

.35: si posiziona sulla 13-esima riga del video e stampa A$ con PRINT#3.

.37: stampa con PRINT#3, sulla linea seguente il valore del numero letto.

.40: chiude il file con lfn=3 e si ferma.

Come vedi in questo esempio il video e' stato aperto come file ed usato alternativamente in Input e in Output. Non si puo', pero', usare l'istruzione INPUT#3, infatti da' l'errore "string too long", dato che non trova il carattere CHR$(13).

1.4 I FILE SU CASSETTA

Nel primo volume abbiamo gia' trattato diffusamente l'argomento dei file su cassetta e riportato un programma per la gestione di un archivio di dati.

Il buffer usato dal sistema per i dati da scrivere o leggere si trova dal byte 819 al byte 1010 (0333H/03F2H); sono 192 byte. La gestione del buffer risulta trasparente per l'utente, che tratta i dati solo a livello logico. Se desideri vedere come i dati sono registrati nel buffer, per esempio dopo la OPEN

di un file, o in qualunque altro momento, puoi leggere il contenuto del buffer usando la funzione PEEK. Inoltre se vuoi vedere i dati presenti nel buffer carattere per carattere li puoi leggere con l'istruzione GET#.
Anche i file di programma transitano dal buffer sia in fase di lettura (LOAD), che in fase di scrittura (SAVE). Risulta meno agevole analizzare il contenuto del buffer per queste operazioni, dato che esse sono svolte con continuita' e, alla fine, nel buffer si trova solo l'ultima parte dei dati transitati.

1.5 I JOYSTICK

Il BASIC ci mette a disposizione un'istruzione per "leggere" lo stato dei Joystick: JOY(n), con n che puo' essere 1 o 2, e corrisponde alle prese JOY 1 e JOY 2.
Questa istruzione puo' essere usata per trasferire lo stato attuale del Joystick in una variabile, scrivendo, per esempio, A=JOY(1). Oppure direttamente, per analizzare in tempo reale lo stato del Joystick, per esempio cosi': IF JOY(1)=1 THEN....
I valori forniti dall'istruzione JOY(n) sono 9 + 9; essi corrispondono agli 8 movimenti possibili + lo stato di riposo, ottenuti con il solo movimento della leva, e agli 8 movimenti possibili + lo stato di riposo con l'aggiunta del fuoco, ottenuti premendo il bottone rosso mentre si muove la leva. Lo schema che segue mostra le due situazioni possibili.

```
          1                            129
      8       2                  136           130
  7       0       3       135          128           131
      6       4                  134           132
          5                            133

    SOLO MOVIMENTO              MOVIMENTO + FUOCO
```

Per utilizzare i joystick come periferiche di Input e' necessario leggere il loro stato e produrre in conseguenza sul video delle azioni, che in generale sono di movimento, ma potrebbero essere di qualunque tipo, infatti dipendono dal programma.

Abbiamo preparato il programma JOYSTICK1 per dimostrare la gestione dei joystick in BASIC; esso gestisce il joystick 1. In esso abbiamo creato dei cicli di attesa per consentire di analizzare gli effetti prodotti. Il programma lavora con il video in modo testo. Inizialmente viene disegnato un quadratino nero in posizione 14,14. Le posizioni corrispondenti (primo numero per riga, secondo per colonna) ai possibili movimenti sono:

13,13	13,14	13,15
14,13	14,14	14,15
15,13	15,14	15,15

```
1 REM JOYSTICK1
4 REM STRINGHE PER MOVIMENTO
7 CG$="█████████████████████"
10 RD$="██████████████████████"
13 R3$=LEFT$(RD$,13):R4$=LEFT$(RD$,14)
16 R5$=LEFT$(RD$,15)
19 C3$=LEFT$(CG$,13):C4$=LEFT$(CG$,14)
22 M$="1234567890"
25 C5$=LEFT$(CG$,15)
28 REM SPAZIO INVERSO E DIRETTO
31 S$="█ █":N$="  "
34 REM NUMERAZIONE RIGHE E COLONNE
37 PRINT"█"M$;M$;M$;M$
40 PRINT"█":FORK=2TO10:PRINT"██";K:NEXTK
43 FORK=1TO10:PRINT"██";K:NEXTK
46 FORK=1TO4:PRINT"██";K:NEXTK:PRINT"██"5;
49 REM QUADRATINO A RIGA 14, COLONNA 14
52 PRINT"█";C4$;R4$;"█";S$;
55 REM LETTURA JOYSTICK 1
58 A=JOY(1):IFA=0THEN58
61 FORK=1TO300:NEXTK
64 PRINT"█";C4$;R4$;N$;:REM CANCELLA
67 IFA=128THEN148
70 IFA>128THEN112
73 REM SOLO MOVIMENTO
76 ONAGOTO82,85,88,91,94,97,100,103
79 STOP
82 PRINT"█";C3$;R4$;"█";S$;:GOTO109
85 PRINT"█";C3$;R5$;"█";S$;:GOTO109
88 PRINT"█";C4$;R5$;"█";S$;:GOTO109
91 PRINT"█";C5$;R5$;"█";S$;:GOTO109
94 PRINT"█";C5$;R4$;"█";S$;:GOTO109
97 PRINT"█";C5$;R3$;"█";S$;:GOTO109
```

```
100 PRINT"▓";C4$;R3$;"■";S$;:GOTO109
103 PRINT"▓";C3$;R3$;"■";S$;:GOTO109
106 REM CICLO DI ATTESA, POI RICOMINCIA
109 FORK=1TO300:NEXTK:A=0:GOTO37
112 REM MOVIMENTO + FUOCO
115 A=A-128
118 ONAGOTO124,127,130,133,136,139,142,145
121 STOP
124 PRINT"▓";C3$;R4$;"▓";S$;:GOTO109
127 PRINT"▓";C3$;R5$;"▓";S$;:GOTO109
130 PRINT"▓";C4$;R5$;"▓";S$;:GOTO109
133 PRINT"▓";C5$;R5$;"▓";S$;:GOTO109
136 PRINT"▓";C5$;R4$;"▓";S$;:GOTO109
139 PRINT"▓";C5$;R3$;"▓";S$;:GOTO109
142 PRINT"▓";C4$;R3$;"▓";S$;:GOTO109
145 PRINT"▓";C3$;R3$;"▓";S$;:GOTO109
148 REM SOLO FUOCO
151 PRINT"▓";C4$;R4$;"▓";S$;:GOTO109
```

COMMENTO A JOYSTICK1

.1/25: prepara stringhe di caratteri di controllo per spostarsi sul video verso il basso e verso destra. R3$ manda a destra di 13 posizioni. R4$ manda a destra di 14 posizioni. R5$ manda a destra di 15 posizioni. C3$ manda in giu' di 13 posizioni. C4$ manda in giu' di 14 posizioni. C5$ manda in giu' di 15 posizioni. M$ serve per numerare le colonne del video.

.28/31: S$ serve per visualizzare uno spazio in campo inverso, N$ per visualizzare uno spazio e quindi cancellare quanto presente in quella posizione.

.34/46: numera le righe e le colonne del video.

.49/52: visualizza un quadratino nero nella posizione 14,14.

.55/64: legge lo stato del joystick 1 fino a quando lo trova diverso da 0, crea un ciclo di attesa, e cancella il quadratino dalla vecchia posizione.

.67/70: se stato=128 va alla linea 148; se stato>128 va alla linea 112.

.73/103: in base al valore letto esegue il movimento, cioe' disegna il quadratino nella nuova posizione in nero.

.106/109: crea un ciclo di attesa e poi torna alla linea 37 per ricominciare.

.112/145: in base al valore letto esegue il movimento, cioe' disegna il quadratino nella nuova posizione in rosso, per segnalare che e' stato premuto anche il bottone del fuoco.

.148/151: disegna nella vecchia posizione il quadratino in rosso, per segnalare solo bottone del fuoco.

Abbiamo preparato il programma JOYSTICK2 per dimostrare la gestione dei joystick in BASIC con video grafico; esso lavora con il joystick 1. In esso abbiamo preparato un piccolo rombo e lo abbiamo memorizzato come SHAPE nella stringa D$. Lo stato del joystick 1 viene usato per spostare lo SHAPE sul video. Anche in questo caso abbiamo creato dei cicli di attesa per consentire di studiare le caratteristiche dei joystick. Se viene premuto anche il bottone del fuoco il rombo viene colorato di nero nella vecchia posizione, poi viene cancellato e disegnato nella nuova posizione. La posizione iniziale e' X1=154, Y1=99. Gli incrementi per X1 e Y1 in base ai movimenti (primo numero per X, secondo per Y) sono i seguenti:

-10,-10	0,-10	10,-10
-10, 0		10, 0
-10, 10	0, 10	10, 10

```
1 REM JOYSTICK2
4 REM ATTIVA MODO GRAFICO
7 GRAPHIC1,1
10 REM PREPARA SHAPE, PICCOLO ROMBO
13 X=159:Y=99
16 DRAW1,X,Y TO X+5,Y+5 TO X,Y+10
19 DRAW1 TO X-5,Y+5 TO X,Y
22 SSHAPED$,X-5,Y,X+5,Y+10
25 X1=X-5:Y1=Y
28 REM INTERROGA JOYSTICK 1
31 A=JOY(1)
34 IFA=0THEN31
37 IFA=128THEN85
40 IFA>128THEN97
43 REM USO JOYSTICK SENZA FUOCO
46 REM XD E YD INCREMENTI COORDINATE
49 IFA=1THENXD=0:YD=-10:GOTO76
52 IFA=2THENXD=10:YD=-10:GOTO76
55 IFA=3THENXD=10:YD=0:GOTO76
58 IFA=4THENXD=10:YD=10:GOTO76
61 IFA=5THENXD=0:YD=10:GOTO76
64 IFA=6THENXD=-10:YD=10:GOTO76
```

```
67 IFA=7THENXD=-10:YD=0:GOTO76
70 IFA=8THENXD=-10:YD=-10:GOTO76
73 REM CANCELLA VECCHIO E DISEGNA NUOVO
76 FORK=1TO300:NEXTK:GSHAPED$,X1,Y1,4
79 X1=X1+XD:Y1=Y1+YD
82 GSHAPED$,X1,Y1:FORK=1TO300:NEXTK:GOTO31
85 REM SOLO FUOCO, COLORA VECCHIO
88 REM CANCELLA E DISEGNA VECCHIO
91 PAINT,X1+5,Y1+5:FORK=1TO300:NEXTK
94 GSHAPED$,X1,Y1:GOTO31
97 REM MOVIMENTO + FUOCO
100 REM COLORA VECCHIO E POI CANCELLA
103 REM DISEGNA NUOVO
106 PAINT,X1+5,Y1+5:FORK=1TO300:NEXTK
109 GSHAPED$,X1,Y1
112 IFA=129THENXD=0:YD=-10:GOTO76
115 IFA=130THENXD=10:YD=-10:GOTO76
118 IFA=131THENXD=10:YD=0:GOTO76
121 IFA=132THENXD=10:YD=10:GOTO76
124 IFA=133THENXD=0:YD=10:GOTO76
127 IFA=134THENXD=-10:YD=10:GOTO76
130 IFA=135THENXD=-10:YD=0:GOTO76
133 IFA=136THENXD=-10:YD=-10:GOTO76
```

COMMENTO A JOYSTICK2

1/7: passa in modo grafico e cancella il video.

10/25: prepara il disegno e lo memorizza. Le coordinate della prima posizione sono X1=154 e Y1=99, angolo in alto a sinistra del rombo.

28/40: memorizza lo stato del joystick 1 e sceglie in base al valore da dove proseguire.

43/70: caso solo movimento: in base al valore letto prepara XD e YD, incrementi per X1 e Y1, per definire la nuova posizione e prosegue.

73/82: crea un ciclo di attesa, cancella vecchio disegno, incrementa X1 e Y1 e disegna nella nuova posizione, poi torna a leggere lo stato del joystick 1.

85/94: caso solo fuoco: colora il rombo nella vecchia posizione, crea un ciclo di attesa, lo cancella e lo ridisegna.

97/133: caso movimento + fuoco: colora il rombo nella vecchia posizione, crea un ciclo di attesa, ridisegna nella vecchia posizione, poi prepara XD e YD, e va alla linea 76.

CAPITOLO 2

I FILE SU STAMPANTE

2.1 INTRODUZIONE

In questo capitolo descriviamo l'uso della stampante COMMODORE MPS-803, che viene di norma venduta per il calcolatore COMMODORE 16. Il calcolatore puo' essere collegato anche ad altre stampanti senza problemi; alcune si possono collegare direttamente, per altre e' necessaria un'interfaccia speciale. La maggior parte dei discorsi che facciamo restano validi anche per altri tipi di stampanti; nel caso devi chiederne al venditore le caratteristiche, vedere se e' necessaria un'interfaccia e leggere accuratamente il manuale allegato per scoprire le differenze.

Figura 2.1 Stampante MPS-803

La MPS-803 e' una stampante a impatto a matrice di punti, nella quale un carattere e' formato da 6 punti orizzontali e 7 punti verticali. Essa riconosce e stampa tutti i caratteri dei due set disponibili sul COMMODORE 16: maiuscolo/grafico e minuscolo/maiuscolo. Inoltre puo' stampare colonne di punti (7 punti verticali) e quindi e' una stampante grafica; la colonna di punti deve essere opportunamente codificata.
Essa si collega tramite l'interfaccia seriale standard al COMMODORE 16, mediante il cavo fornito che termina con uno spinotto DIN a 6-pin, dove i pin hanno il significato riportato nella Figura 2.2.

CONNETTORE

Pin No.	Signal
1	SERIAL SRQ
2	GND
3	SERIAL ATN
4	SERIAL CLK
5	SERIAL DATA
6	RESET

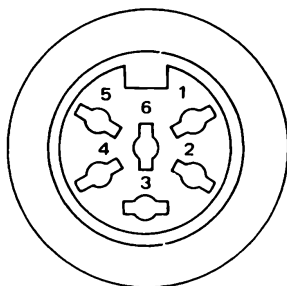



Figura 2.2 Presa di collegamento per la MPS-803

Sulla parte superiore della MPS-803 sono visibili:
.in alto a sinistra, la manopola per l'avanzamento manuale della carta;
.in alto a destra, la levetta per il bloccaggio della carta (da usare quando non e' montato il trascina moduli);
.in basso a destra, una spia luminosa rossa, marcata Power, che segnala:
.con luce continua che la macchina e' accesa,
.con luce intermittente che si e' verificato un errore,
e vicino un tasto a pressione, marcato Paper Advance, per l'avanzamento della carta.
La copetura anteriore puo' essere sollevata, agendo su

due appositi incavi laterali. Per inserire il nastro inchiostrato si deve sollevare tale copertura e si rende visibile l'alloggiamento del nastro. Nel manuale allegato alla stampante sono riportate delle illustrazioni che insegnano a montare il nastro.

Sul lato destro in basso si trova l'interruttore di accensione.

La stampante puo' essere alimentata con moduli continui o con fogli singoli, usando la levetta di bloccaggio della carta (quando e' in posizione OPEN la carta e' libera); oppure puo' essere montato il trascina moduli, richiedendolo al venditore. La larghezza della linea di stampa e' di 80 caratteri (larghi 6 punti), quindi di 480 punti. Possiamo assumere la larghezza della linea di stampa come dimensione del record fisico. La stampa e' bidirezionale e la velocita' di stampa e' di 60 caratteri al secondo. Si possono stampare fino a 3 copie. La stampante non puo' funzionare se la carta non e' inserita. Quando, durante la stampa, termina la carta, comincia a pulsare la spia rossa Power e la stampa si interrompe; per proseguire devi inserire nuova carta e premere il tasto Paper Advance.

Nella parte posteriore vediamo quanto riportato nella Figura 2.3.

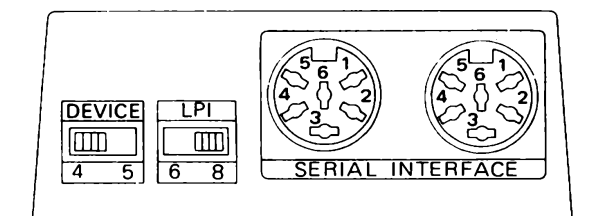



Figura 2.3 Parte posteriore della stampante MPS-803

Lo switch DEVICE puo' essere posto nella posizione 4 o nella posizione 5, dando la possibilita' di scegliere il "dn" della stampante. Di solito si usa il 4, ma, se sono collegate contemporaneamente due stampanti, una deve avere dn=5.
Lo switch LPI puo' essere posto nelle posizioni 6 o 7; esso indica la distanza di 1/6 o 1/8 di pollice tra due linee di stampa.
La porta seriale SERIAL INTERFACE dispone di due ingressi; uno serve per collegare la stampante al calcolatore, l'altro per collegare in cascata una seconda stampante o una unita' a floppy disk. Si possono collegare in cascata piu' periferiche (vedi Paragrafo 4.10). Se al calcolatore e' collegata una unita' a floppy disk, la MPS-803 si collega ad essa.

Il COMMODORE 16 puo' inviare dati alla stampante dopo avere stabilito una comunicazione con essa; la stampante riceve un flusso di dati e stampa un file. La MPS-803 contiene un microprocessore che gestisce le operazioni di stampa, eseguendo apposite routine memorizzate in una ROM interna, servendosi di una descrizione dei caratteri memorizzata in una ROM interna, e di un buffer di stampa, cioe' di una parte di RAM interna.
Il COMMODORE 16 invia i dati, attraverso l'interfaccia seriale, un bit dopo l'altro; ogni gruppo di 8 bit (un byte) contiene un codice ASCII, che puo' variare da 0 a 255. I byte sono ricevuti e memorizzati uno dopo l'altro nel buffer, che puo' contenere fino a 90 byte; vedremo piu' avanti in quali circostanze avviene realmente la stampa.
Alcuni dei codici inviati non sono caratteri da stampare, ma codici di controllo che agiscono sul modo di funzionare della stampante.
Quando la stampante funziona, al momento dell'accensione, sia che sia collegata al calcolatore, sia che non lo sia, la testina di stampa viene spostata dal centro verso sinistra e poi riportata al centro.
Per vedere se la stampa e' corretta puoi farle eseguire l'auto-test. Devi procedere cosi':

.aver montato correttamente il nastro e inserito la carta,

.dare corrente alla stampante, anche non collegata al calcolatore, mentre tieni premuto il tasto anteriore "Paper Advance", poi rilasciare tale tasto.

La stampante stampa ripetutamente tutti i caratteri

stampabili. Per interrompere il test devi togliere corrente.

2.2 ISTRUZIONI DI STAMPA

Prima di stampare si deve aprire la comunicazione tra il calcolatore e la stampante con l'istruzione OPEN; essa si scrive:

```
OPEN lfn,dn,sa
```

.lfn (Logical File Number), e' il numero logico del file che si vuole aprire e che deve essere usato nelle istruzioni di stampa successive; esso puo' variare da 0 a 255. Se lfn>127 si ottiene una spaziatura doppia tra una linea e la successiva.
.dn (Device Number), e' il numero della periferica, puo' essere 4 o 5, e deve essere quello predisposto dall'apposito switch posto sul retro della stampante.
.sa (Secondary Address), e' un numero che puo' valere 0 (valore di default) o 7 e determina il set di caratteri che la stampante deve usare nelle successive operazioni di stampa:
    .0, o niente, per il set maiuscolo/grafico,
    .7, per il set minuscolo/maiuscolo.
I parametri possono essere costanti o variabili con valore intero.
Il set di caratteri scelto con la OPEN resta attivo fino alla CLOSE; si puo' usare all'interno della lista di stampa un codice di controllo che modifica temporaneamente il set, ma esso e' valido solo per la stampa in corso, cioe' fino al primo carattere RETURN.

L'istruzione di stampa e':

```
PRINT# lfn,lista
```

.lfn, deve essere lo stesso usato nella OPEN.
.lista, e' l'insieme dei dati da stampare, dei caratteri separatori, delle funzioni di stampa e dei codici di controllo per la stampante. Negli esempi successivi vedremo le caratteristiche di ogni elemento che puo' comparire nella lista.
Per scrivere questa istruzione non si puo' usare il "?" per abbreviare la parola chiave, inoltre non deve esserci spazio prima del carattere "#".

Il calcolatore, dopo aver eseguito un'istruzione PRINT#, chiude la comunicazione con la stampante, ma il file logico rimane aperto; esso viene chiuso solo dall'istruzione CLOSE.

Si puo' stampare su carta, anche usando l'istruzione CMD, dopo aver aperto un file per la stampante. Questa istruzione trasferisce l'uscita, che normalmente avviene sul video, alla stampante. Si scrive:

```
CMD lfn,lista
```

.lfn, deve essere lo stesso usato nell'istruzione OPEN.
.lista, puo' essere una normale lista di stampa o mancare.

Dopo l'esecuzione di CMD, con o senza "lista", le successive istruzioni PRINT (scritte senza il carattere "#") mandano l'output sulla stampante. Per far terminare l'effetto di CMD, cioe' il "dirottamento" dell'uscita dal video a un'altra periferica, e' necessario eseguire una PRINT#lfn, con "lfn" uguale a quello usato per CMD, che serve per chiudere la linea, prima della CLOSE. In caso contrario non si ha un funzionamento corretto del sistema; inoltre, quando si e' in stato CMD, non si deve accedere a un'altra periferica collegata in serie (disco o altra stampante). Il comando LIST, usato dopo CMD, provoca la lista del programma sulla stampante.
Il modo corretto per ottenere la lista dei programmi sulla stampante e' eseguire in immmediato:

```
OPEN4,4:CMD4:LIST:PRINT#4:CLOSE4
```

puo' essere utile assegnare a un tasto funzione la sequenza di istruzioni necessarie per listare i programmi, eseguendo per esempio:
KEY1,"OPEN4,4:CMD4:LIST:PRINT#4:CLOSE4"+CHR$(13)
in modo che premendo F1 si ottiene la lista con chiusura corretta del file.

Quando le operazioni di stampa sono terminate, deve essere eseguita l'istruzione:

```
CLOSE lfn
```

che serve per chiudere la comunicazione. Dopo l'ese-

cuzione della CLOSE non si puo' piu' comunicare con il file "lfn", se non si esegue nuovamente una OPEN.

E' importante non lasciare aperti i file che non servono piu'; si rischia di occupare inutilmente posto nella tabella dei file, che ha solo 10 posti.

Le istruzioni di stampa possono essere usate sia in modo immediato che da programma.

ESEMPI DI STAMPA IN MODO IMMEDIATO

Usando in modo immediato le 4 sequenze di istruzioni che seguono, si ottiene sempre lo stesso risultato sulla stampante e si opera correttamente chiudendo sia la linea che il file logico.

.1)
```
OPEN4,4:PRINT#4,"MPS-803":CLOSE4
```
stampa MPS-803 e va a capo.

.2)
```
OPEN4,4:CMD4,"MPS-803":PRINT#4:CLOSE4
```
stampa MPS-803 e va a capo.

.3)
```
OPEN4,4:CMD4:PRINT"MPS-803":PRINT#4:CLOSE4
```
stampa MPS-803 e va a capo.

.4)
```
OPEN4,4,7:CMD4:PRINT"MPS-803":PRINT#4:CLOSE4
```
stampa mps-803 (abbiamo usato sa=7) e va a capo.

ESEMPI DI STAMPA DA PROGRAMMA

Il programma SAOCMDLIST, che segue con i suoi risultati, apre il file logico con lfn=1 sulla stampante (dn=4), usando sa=0, che poteva anche essere saltato, dato che 0 e' il valore di default, e quindi lavora con il set maiuscolo/grafico. Nel programma alla linea 15 e' inserito il comando LIST, cioe' il programma lista se stesso, poi continua e stampa una frase. Alla linea 25 viene eseguita l'istruzione PRINT#1 per chiudere la linea e poi la CLOSE1.

```
1 REM SA0CMDLIST
5 OPEN1,4,0:REM APRE CON SA=0
10 CMD1:REM USCITA DA VIDEO A STAMPANTE
15 LIST:PRINT
20 PRINT"OPEN CON SA=0 E USO CMD E LIST"
25 PRINT#1:CLOSE1

OPEN CON SA=0 E USO CMD E LIST
```

Il programma SA7CMDLIST e' uguale al precedente, salvo che apre il file di stampa con sa=7 e quindi lavora con il set di caratteri minuscolo/maiuscolo.

```
1 rem sa7cmdlist
5 open1,4,7:rem apre con sa=7
10 cmd1:rem uscita da video a stampante
15 list:print
20 print"open con sa=7 e uso cmd e list"
25 print#1:close1

open con sa=7 e uso cmd e list
```

Segue il programma SETMPS-803, che stampa in forma tabellare i due set di caratteri disponibili sulla stampante. Il disegno dei caratteri e' quello registrato nella ROM interna alla stampante, che differisce un po' dalla forma dei caratteri che appare sul video: 6x7 punti per la stampante, 8x8 punti sul video.

```
1 REM SETMPS-803
5 A$="       SET MAIUSCOLO/GRAFICO"
6 SA=0
10 OPEN4,4,SA:PRINT#4,A$
11 PRINT#4:PRINT#4," | ";
12 FORK=0TO9:PRINT#4,MID$(STR$(K),2);" ";
13 NEXTK
14 FORK=65TO70:PRINT#4,CHR$(K);" ";:NEXTK
15 PRINT#4:PRINT#4,CHR$(192)"┤";
16 FORK=1TO16:PRINT#4,CHR$(192)CHR$(192);
17 NEXTK:PRINT#4:I=0
19 FORK=0TO9:PRINT#4,MID$(STR$(K),2);"| ";
20 GOSUB100:I=I+1:NEXTK
23 FORK=65TO70:PRINT#4,CHR$(K);"| ";
25 GOSUB100:I=I+1:NEXTK
```

```
27 IFSA=7THENCLOSE4:STOP
29 PRINT#4:PRINT#4:SA=7
30 A$="       SET MAIUSCOLO/MINUSCOLO"
31 CLOSE4:GOTO10
100 IK=I:FORL=1TO16
101 IFIK<32THENPRINT#4," ";:GOTO104
102 IFIK>127ANDIK<160THENPRINT#4," ";:GOTO104
103 PRINT#4,CHR$(IK);" ";
104 IK=IK+16:NEXTL:PRINT#4:RETURN
```

I caratteri da stampare sono inviati come codici ASCII; il programma stampa spazi al posto dei caratteri corrispondenti ai codici da 0 a 31 e da 128 a 159, formando le due colonne vuote nelle tabelle, dato che non corrispondono a caratteri stampabili. Le coordinate orizzontali e verticali delle due tabelle sono espresse in esadecimale (le cifre esadecimali da A a F corrispondono ai numeri decimali da 10 a 15).

Per ottenere il codice ASCII dei caratteri devi moltiplicare per 16 la coordinata di colonna (sopra) e aggiungere la coordinata di riga (laterale).

RISULTATI PROGRAMMA SETMPS-803

SET MAIUSCOLO/GRAFICO

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0				0	@	P	─	┐				┌	─	┐		┌
1			!	1	A	Q	♠	●			▌	┴	♠	●	▌	┴
2			"	2	B	R	│	─			▄	┬	│	─	▄	┬
3			#	3	C	S	─	♥			▔	┤	─	♥	▔	┤
4			$	4	D	T	─	│			▁	│	─	│	▁	│
5			%	5	E	U	─	╭			▏	▌	─	╭	▏	▌
6			&	6	F	V	─	╳			▒	▐	─	╳	▒	▐
7			'	7	G	W	│	○			▕	▔	│	○	▕	▔
8			(	8	H	X	│	♣			▒	▀	│	♣	▒	▀
9			)	9	I	Y	╮	│			◤	▃	╮	│	◤	▃
A			*	:	J	Z	╰	♦			▕	┘	╰	♦	▕	┘
B			+	;	K	[	╯	┼			├	▘	╯	┼	├	▘
C			,	<	L	£	L	▒			▗	▝	L	▒	▗	▝
D			-	=	M	]	╲	│			└	┘	╲	│	└	┘
E			.	>	N	↑	╱	π			┐	▀	╱	π	┐	▀
F			/	?	O	←	┌	◥			▂	▚	┌	◥	▂	π

set maiuscolo/minuscolo

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0				0	@	p	─	P				┌	─	P		┌
1			!	1	a	q	A	Q			▌	┴	A	Q	▌	┴
2			"	2	b	r	B	R			▄	┬	B	R	▄	┬
3			#	3	c	s	C	S			▔	┤	C	S	▔	┤
4			$	4	d	t	D	T			▁	▎	D	T	▁	▎
5			%	5	e	u	E	U			▏	▍	E	U	▏	▍
6			&	6	f	v	F	V			▒	▐	F	V	▒	▐
7			'	7	g	w	G	W			▕	▔	G	W	▕	▔
8			(	8	h	x	H	X			▒	▀	H	X	▒	▀
9			)	9	i	y	I	Y			◤	▃	I	Y	◤	▃
a			*	:	j	z	J	Z			▕	✓	J	Z	▕	✓
b			+	;	k	[	K	┼			├	▖	K	┼	├	▖
c			,	<	l	£	L	▒			▗	▝	L	▒	▗	▝
d			-	=	m	]	M	│			└	┘	M	│	└	┘
e			.	>	n	↑	N	✕			┐	▘	N	✕	┐	▘
f			/	?	o	←	O	▒			▂	▚	O	▒	▂	▚

COMMENTO A SETMPS-803

.5/6: prepara in A$ l'intestazione della prima tabella e pone SA=0.
.10: apre il file logico con lfn=4, il valore attuale di SA per la stampante, e stampa il titolo della tabellina.
.11/17: stampa le coordinate di colonna e le linee separatrici (CHR$(192) da' luogo a un trattino), e pone il contatore I a 0.
.19/20: stampa le linee corrispondenti alle coordinate di riga da 0 a 9, servendosi del sottoprogramma in 100 per stampare i caratteri del set attivo.
.23/25: come sopra per le linee con cordinate da A a F.
.27: se SA=7 chiude il file logico e si ferma.
.29/31: pone SA=7, prepara la nuova intestazione in A$, chiude il file logico e torna alla linea 10 per stampare la seconda tabellina.
.100/104: sottoprogramma che in base al codice del carattere, che trova inizialmente in I e pone in IK, stampa, se sono stampabili, i 16 caratteri di una linea. Non stampa il carattere se il codice e' minore di 32 oppure compreso tra 128 e 159, estremi inclusi. I caratteri che stanno su una linea hanno codici che differiscono di 16 da quello precedente.

2.3 MODI DI STAMPA E USO DEI CARATTERI DI CONTROLLO

Il valore di "sa" usato nella OPEN predispone l'uso di uno dei due set di caratteri disponibili nella MPS-803, in modo permanente, fino all'esecuzione della relativa CLOSE. Il valore di "sa" influenza la stampa in MODO CARATTERE, cioe' quella attiva al momento dell'accensione della stampante. Per ottenere il passaggio al MODO GRAFICO devi inviare un carattere di controllo (CHR$(8)); in tale caso e' necessario inviare un altro carattere di controllo per tornare al modo testo (CHR$(15)).
Possiamo considerare come una prima distinzione nelle possibilita' di stampa i due modi: carattere e grafico; ad essi possiamo aggiungere gli altri modi che si ottengono con l'uso di alcuni caratteri di controllo.
Segue l'elenco dei caratteri di controllo che hanno un particolare significato per la stampante MPS-803; essi devono essere inviati nella "lista" di un'istruzione PRINT diretta alla stampante, come stringa, usando la funzione CHR$. Per ogni carattere indichiamo: il codice, il significato, i parametri che devono seguire il codice, se necessario, e il numero dei byte che vengono occupati in conseguenza nel buffer della stampante. Se un codice richiede dei parametri, ovviamente questi vengono trasmessi, ma non stampati. Alcuni codici vanno usati insieme ad altri, altrimenti perdono significato.
Tieni presente che i codici di controllo hanno in generale significato diverso se diretti al video invece che alla stampante; gli unici che producono lo stesso effetto sono 10, 13, 18 e 146.

CODICE	SIGNIFICATO	PARAMETRI	NUM. BYTE
8	MODO GRAFICO	no	1
10	invio LINE FEED e RETURN	no	1
13	invio LINE FEED e RETURN	no	1
14	MODO CARATTERE ALLLARGATO	no	1
15	MODO CARATTERE NORMALE	no	1
16	SPOSTAMENTO POSIZIONE STAMPA A UNA COLONNA	2	3

17	PASSAGGIO SET MINUSC./MAIUSC.	no	1
18	MODO RVS-ON	no	1
26	MODO RIPETIZIONE CAR. GRAFICO	1	2
27	SPOSTAMENTO POSIZIONE GRAFICA deve essere seguito da CHR$(16)	2	4
145	PASSAGGIO SET MAIUSC./GRAFICO	no	1
146	MODO RVS-OFF	no	1

Esaminiamo dettagliatamente i singoli codici di controllo, riportando alcuni programmi esempio. In alcuni programmi esempio abbiamo usato la tecnica di far lavorare il programma stampando i risultati, poi il programma lista se stesso; in conseguenza, in questi casi, vedrai prima i risultati e poi il listato del programma.

==

CHR$(8)   MODO GRAFICO

Predispone la stampa in modo grafico; tale modo resta attivo fino all'invio dei codici di controllo 14 o 15, che riportano rispettivamente in modo carattere allargato e in modo carattere normale.
Dopo aver attivato il modo grafico devi passare nella lista di stampa i codici dei caratteri grafici da stampare, come stringa. Ogni carattere grafico e' formato da una colonna di 7 punti, e viene codificato secondo le seguenti regole:

.i punti da disegnare devono corrispondere alla cifra 1, quelli da non disegnare devono corrispondere alla cifra 0,

.le 7 linee hanno pesi diversi, partendo dall'alto i pesi sono: 1, 2, 4, 8, 16, 32, 64, cioe' le potenze di 2, partendo dall'esponente 0 e arrivando all'esponente 6,

.devi moltiplicare ogni cifra per il peso corrispondente e sommare i valori, per una colonna con 7 cifre 1 ottieni 1+2+4+8+16+32+64=127, per una colonna con 7 cifre 0 ottieni 0.

.al numero ottenuto devi aggiungere 128, in tale modo il codice calcolato puo' variare da 128 a 255.
Il codice deve essere passato nella "lista" di stampa con la funzione CHR$. Se desideri ottenere un disegno composto da piu' colonne, devi ripetere il procedimento spiegato per ogni colonna.

Abbiamo preparato il disegno di un omino con le braccia alzate, occupando 11 colonne di 7 punti; lo riportiamo indicando con asterischi i punti da disegnare e con lineette quelli da lasciare in bianco. Inoltre riportiamo di fianco l'immagine ottenuta con 0 e 1 e con a margine i pesi di ogni punto e sotto ogni colonna il codice risultante dal calcolo.

```
-**-***-**-     1)  0  1  1   0  1  1  1   0  1  1  0
--**-*-**--     2)  0  0  1   1  0  1  0   1  1  0  0
---*****---     4)  0  0  0   1  1  1  1   1  0  0  0
----***----     8)  0  0  0   0  1  1  1   0  0  0  0
----***----    16)  0  0  0   0  1  1  1   0  0  0  0
---**-**---    32)  0  0  0   1  1  0  1   1  0  0  0
-***---***-    64)  0  1  1   1  0  0  0   1  1  1  0

Valore codici:      0 65 67 102 61 31 61 102 67 65  0
```

Aggiungendo 128 agli 11 codici otteniamo:
128, 193, 195, 230, 189, 159, 189, 230, 195, 193, 128, passando dopo CHR$(8) le funzioni CHR$ degli 11 codici otteniamo il disegno di un omino. Il nostro disegno e' simmetrico e inizia e finisce con una colonna bianca; per questo disegnando vicino alcuni omini essi non risultano attaccati insieme.

Nel programma COD8-1 abbiamo inserito gli 11 numeri con una linea DATA, la 4; poi prepariamo nella stringa A$ l'omino completo come stringa; stampando A$, in modo grafico, otteniamo il disegno.

```
XXXXX
XXXXXXX
XXXXXXXXX
XXXXXXXXXXX
```

```
1 rem cod8-1
2 open10,4,7
3 cmd10
4 data0,65,67,102,61,31,61,102,67,65,0
5 a$="":forj=1to11
6 reada:a=a+128
7 a$=a$+chr$(a)
8 nextj
9 fork=1to5
10 printchr$(8)a$;
11 nextk:print:print
12 fork=1to7
13 printa$;
14 nextk:print:print
15 fork=1to9
16 printa$;
17 nextk:print:print
18 fork=1to11
19 printa$;
20 nextk:print
21 printchr$(15)
22 print:list:print#10:close10
```

COMMENTO A COD8-1

.2/3: apre la stampante con lfn=10 e sa=7, con CMD10 trasferisce l'uscita video alla stampante.
.4: linea DATA con i codici delle 11 colonne di punti, prima di aggiungere 128.
.5/8: legge gli 11 codici, aggiunge 128 a ogni codice e costruisce la stringa A$ con il disegno.
.9/11: stampa 5 omini, dopo aver attivato la stampa grafica con CHR$(8). Il codice e' usato all'interno del ciclo FOR, e quindi viene inviato 5 volte; bastava inviarlo una sola volta prima del ciclo FOR. Alla linea 11 esegue 2 volte PRINT, la prima volta per andare a capo e la seconda per produrre uno spazio. In modo grafico la distanza tra le righe e' minore rispetto al modo carattere.

.12/14: stampa 7 omini, lavorando come sopra, ma senza inviare di nuovo il codice 8.
.15/17: stampa 9 omini.
.18/20: stampa 11 omini.
.21: ritorna in modo carattere normale stampando CHR$(15)
.22: lista se stesso, chiude la linea e il file.

Nel programma esempio COD8-2, abbiamo disegnato una casetta che occupa un rettangolo di 17 colonne e 21 righe. Per poterla stampare la dobbiamo dividere a strisce alte 7 punti ciascuna; otteniamo 3 strisce. Stamperemo le tre parti una per riga, una sotto l'altra.
Per evitare la fatica di calcolare i codici abbiamo disegnato la casetta in 21 linee di programma, 21 linee DATA, scritte una dopo l'altra. Prima, con una REM abbiamo numerato le colonne, per disegnare piu' facilmente. La casetta viene disegnata usando gli asterischi.
Abbiamo incorporato nel programma una routine che legge le 21 stringhe, 7 per volta, analizza i caratteri delle stringhe e sostituisce 1 agli asterischi e 0 agli spazi. Poi calcola il valore di ogni colonna, aggiunge 128 e costruisce la stringa con la funzione CHR$ di ogni codice.
Puoi estrarre questa routine e adattarla a disegni di altre dimensioni. Ricordati di usare un numero di righe multiplo di 7.
Il programma stampa 10 volte la casetta e poi lista se stesso.

```
1 REM COD8-2
3 DIMT$(7),C$(3):REM STRINGHE PER DISEGNO
4 DIMT(6):REM PER CALCOLO
8 REM DISEGNO CASETTA IN 21 LINEE DATA
9 REM  12345678901234567
10 DATA"                 "
11 DATA"               * "
12 DATA"               * "
13 DATA"        *     *  "
14 DATA"       * *   *   "
15 DATA"      *   * ***  "
16 DATA"     *     ** *  "
17 DATA"    *       * *  "
18 DATA"   *         **  "
19 DATA"  *           *  "
20 DATA" *************** "
21 DATA" *             * "
22 DATA" * ***     *** * "
23 DATA" * * *     * * * "
24 DATA" * ***     *** * "
25 DATA" *             * "
26 DATA" *             * "
27 DATA" *     ***     * "
28 DATA" *     * *     * "
29 DATA" *     * *     * "
30 DATA" *************** "
36 PRINT"♥STO CALCOLANDO I CODICI DELLA CASETTA"
40 REM CALCOLO COLONNE DI PUNTI
45 FORK=1TO3:C$(K)="":FORJ=1TO7
50 READT$(J):PRINTT$(J):NEXTJ
55 FORL=1TO17:I=0
60 FORJ=1TO7:T(I)=ASC(MID$(T$(J),L,1))
65 IFT(I)=42THENT(I)=1:ELSET(I)=0
70 I=I+1:NEXTJ
75 T=0:FORJ=0TO6:T=T+T(J)*2↑J:NEXTJ:T=T+128
80 C$(K)=C$(K)+CHR$(T):NEXTL
85 NEXTK
100 REM STAMPA CASETTA
105 OPEN4,4:PRINT#4,CHR$(8)
110 FORJ=1TO3:FORK=1TO10
115 PRINT#4,C$(J);:NEXTK:PRINT#4
120 NEXTJ
125 PRINT#4,CHR$(15)
130 CMD4:LIST:PRINT#4:CLOSE4
```

COMMENTO A COD8-2

.3: dimensiona T$(7) per contenere le 7 stringhe di una striscia, e C$(3) per contenere le 3 stringhe da stampare per formare il disegno.

.4: dimensiona un vettore T(6) per contenere i 7 numeri corrispondenti a una colonna di punti.

.8/30: disegno della casetta in linee DATA.

.36: stampa un messaggio per avvisare che esegue il calcolo dei codici.

.40/85: ciclo per calcolare i codici della casetta.

.45: inizia il ciclo per K da 1 a 3 per calcolare le 3 stringhe che costruiscono il disegno, pulisce la stringa e inizia un ciclo per J da 1 a 7 per leggere 7 linee DATA.

.50: legge la stringa, la stampa sul video, disegnando cosi' la casetta con gli asterischi, e chiude il ciclo di J.

.55: inizia il ciclo per analizzare i 17 caratteri di ogni stringa, e pone I=0, indice per il vettore T.

.60/70: preleva da ognuna delle 7 stringhe i 7 caratteri che occupano la stessa posizione verticale e trasforma gli asterischi in 1 e gli spazi in 0 prima di porli nel vettore T(I). Alla fine del ciclo il vettore T(I) contiene una colonna di punti in cifre 1 e 0.

.75: calcola il codice corrispondente moltiplicando le cifre per i relativi pesi, poi aggiunge 128.

.80: aggiunge alla stringa C$(K) il nuovo carattere grafico calcolato e chiude il ciclo di L. Alla fine di questo ciclo la stringa C$(K) contiene tutti i 17 codici della striscia relativa, trasformati in carattere ASCII.

.85: chiude il ciclo di K. Alla fine sono pronte le 3 stringhe C$(K).

.100/120: stampa 10 casette ripetendo 10 volte ogni disegno e poi andando a capo.

.125: disattiva il modo grafico e torna al modo carattere normale.

.130: lista se stesso e chiude.

==

CHR$(10) e CHR$(13)   INVIO LINE FEED E RETURN

Questi due codici hanno lo stesso comportamento, ognuno di essi stampa un LINE FEED e un RETURN e provoca la stampa di tutto quello che e' contenuto nel buf-

fer. Se termini una "lista" con uno di questi codici, e non aggiungi il ";" finale, la mancanza di punteggiatura provoca un ulteriore RETURN.

Nel programma COD10/13 che segue, ti mostriamo come l'effetto dei due codici sia il medesimo e come influisce il valore di "lfn" sulla spaziatura tra le linee.

```
1 REM COD10/13
5 OPEN129,4
10 A$="PROVA1 LFN>128":B$="PROVA DI CHR$(10)"
15 PRINT#129,A$CHR$(10)B$
20 C$="PROVA2 LFN>128":D$="PROVA DI CHR$(13)"
25 PRINT#129,C$CHR$(13)D$
30 CLOSE129
35 OPEN10,4
40 E$="PROVA3 LFN<128":F$="PROVA DI CHR$(10)"
45 PRINT#10,E$CHR$(10)F$
50 G$="PROVA4 LFN<128":H$="PROVA DI CHR$(13)"
55 PRINT#10,G$CHR$(13)H$
60 CLOSE10
65 STOP
```

RISULTATI PROGRAMMA COD10/13

```
PROVA1 LFN>128
PROVA DI CHR$(10)

PROVA2 LFN>128
PROVA DI CHR$(13)

PROVA3 LFN<128
PROVA DI CHR$(10)
PROVA4 LFN<128
PROVA DI CHR$(13)
```

COMMENTO A COD10/13

.5: apre con lfn=129 (>127) e questo provoca una doppia spaziatura a fine linea se manca la punteggiatura finale.
.10/15: prepara le due stringhe A$ e B$ e le stampa separandole con CHR$(10); esse vengono stampate una su ogni riga, ma alla fine si ha un doppio spazio.

.20/25: come sopra, ma separando le due stringhe con CHR$(13); si ottiene lo stesso effetto di prima.
.30: chiude il file con lfn=129.
.35: apre il file con lfn=10 (<127) e questo provoca una spaziatura semplice in assenza di punteggiatura finale.
.40/60: stampa con gli stessi codici di controllo e ottiene la spaziatura semplice anche in assenza di punteggiatura finale.
Se la punteggiatura finale invece di essere un ";" e' una ",", si ha l'aggiunta di 10 spazi, e questo puo' provocare il passaggio a nuova linea, anche se non sono presenti o il codice 10 o il codice 13.

CHR$(14)   MODO CARATTERE ALLARGATO

Predispone la stampa in modo testo del carattere allargato, cioe' del carattere formato da 12 punti per riga e 7 punti per colonna. La predisposizione rimane fino a quando si invia il codice 15, per tornare al carattere normale, o il codice 8 per passare in modo grafico.
Segue il programma COD14-1, che stampa due linee in carattere allargato, poi torna al modo normale, lista se stesso e chiude correttamente la comunicazione.

```
COMMODORE
 MPS-803

1 REM COD14-1
2 OPEN10,4
3 CMD10
4 PRINTCHR$(14)"COMMODORE"
5 PRINTCHR$(14)" MPS-803 "
6 PRINTCHR$(15)
7 LIST
8 PRINT#10:CLOSE10
```

Il programma COD14-2, invece, mostra come si possono ottenere sulle stesse linee di stampa sia caratteri normali che caratteri allargati, usando alternativamente i codici 14 e 15.

```
1 REM COD14-2
2 OPEN10,4
3 PRINT#10,CHR$(14)"COMMODORE ";
4 PRINT#10,CHR$(15)"COMMODORE "
5 PRINT#10,CHR$(14)" MPS-803 ";
6 PRINT#10,CHR$(15)" MPS-803 "
7 FORK=65TO68
8 PRINT#10,CHR$(14)CHR$(K)CHR$(15)CHR$(K);
9 NEXTK
10 PRINT#10,CHR$(15):CLOSE10
```

In questo caso stampiamo con PRINT#10 e non trasferendo la stampa dal video alla stampante con CMD10. Nota alle linee 7/9, come otteniamo in ciclo la stampa alternata di un carattere allargato e uno normale; inoltre scriviamo le variabili stringa una vicino all'altra senza punteggiatura, ma dobbiamo porre un ";" finale per evitare che vada a capo.

RISULTATI PROGRAMMA COD14-2

```
COMMODORE  COMMODORE
 MPS-803   MPS-803
AABBCCDD
```

CHR$(15) MODO CARATTERE NORMALE

Predispone la stampa nel modo normale, che e' attivo all'accensione. Devi usare questo codice per disattivare sia il modo a carattere allargato, che il modo grafico.

CHR$(16) SPOSTAMENTO POSIZIONE STAMPA A UNA COLONNA

Questo codice predispone l'inizio della stampa a una colonna, tra 00 e 79. Il numero della colonna, espresso come stringa di 2 caratteri, deve seguire immediatamente CHR$(16). Ricorda che le cifre numeriche hanno codice ASCII che varia da 48 a 57; per indicare la colonna 18 puoi scrivere "18" oppure CHR$(49)CHR$(56). Il codice 16 puo' essere usato sia in modo testo che in modo grafico. Riportiamo un esempio nel programma COD16-1.

```
0123456789012345678901234567890123456789012345678901234567890123456789
 0 X                    1 X                    2 X
0123456789012345678901234567890123456789012345678901234567890123456789
X 0                    X 1                    X 2

1 REM COD16-1
5 OPEN10,4
10 CMD10:A$=""
15 DATA0,65,67,102,61,31,61,102,67,65,0
20 DATA48,48,49,53,51,48
25 FORI=1TO11:READA
30 A$=A$+CHR$(A+128)
35 NEXTI
40 GOSUB100
50 FORI=0TO2:READX,Y
55 PRINTCHR$(15)CHR$(16)CHR$(X)CHR$(Y);I;CHR$(8)A$;
60 NEXTI:PRINTCHR$(15)
65 GOSUB100:RESTORE20:FORI=0TO2:READX,Y
70 PRINTCHR$(8)CHR$(16)CHR$(X)CHR$(Y);A$;CHR$(15);I;
75 NEXTI
80 PRINTCHR$(15):LIST
85 PRINT#10:CLOSE10:STOP
100 FORK=0TO4:FORI=0TO9:PRINTCHR$(48+I);
105 NEXTI:NEXTK:PRINT:RETURN
```

COMMENTO A COD16-1

.5/10: apre il file logico 10 per la stampante, trasferisce l'uscita video alla stampante e pulisce la stringa A$.

.15: linea DATA che contiene i codici dell'omino con le braccia alzate, precedentemente preparato; esso occupa 11 colonne di punti.

.20: linea DATA che contiene 3 coppie di cifre, in codice ASCII, per definire le tre colonne: 00, 15, 30.

.25/35: preparazione in A$ del disegno dell'omino, aggiungendo 128 a ogni codice e trasformandolo in stringa.

.40: esecuzione del sottoprogramma in 100 per stampare una linea di numerazione delle posizioni di stampa.

.50/60: stampa in ciclo, dopo aver letto le cifre di ogni colonna dalla linea DATA 20, aver definito la posizione di stampa, del numero I e dell'omino. Nota nei risultati che le colonne selezionate sono 00, 15 e 30, che i numeri sono stampati preceduti e seguiti da uno spazio, e che gli omini occupano quasi due posizioni carattere. In questo caso il codice 16 viene usato trovandosi in modo carattere per effetto del

codice 15. Concluso il ciclo, va a capo e ripristina il modo carattere con il codice 15.
.65/75: esegue nuovamente il sottoprogramma per numerare le posizioni di stampa, poi passa con il codice 8 in modo grafico e usa il codice 16 per definire la posizione della colonna di inizio stampa. Nel ciclo viene stampato prima l'omino e poi il numero I, dopo essere tornati in modo carattere con il codice 15. Nota nei risultati che: gli omini iniziano esattamente alle colonne 00, 15 e 30 (la numerazione va da 0 a 79), mentre i numeri iniziano esattamente 8 punti dopo l'omino, cioe' risultano sfalsati rispetto alla sovrastante colonna di numerazione.
.80/85: ripristino del modo carattere, lista del programma e chiusura del file.
.100/105: sottoprogramma di numerazione linea di stampa da 0 a 9 per 5 volte.

CHR$(17) PASSAGGIO AL SET MINUSCOLO/MAIUSCOLO

Questo codice fa passare al set minuscolo/maiuscolo, con validita' locale, cioe' solo per l'istruzione di PRINT in corso (fino al primo RETURN), indipendentemente dal set selezionato con l'istruzione OPEN, che torna attivo al termine della PRINT.
Osserva piu' avanti i 4 programmi esempio da COD17/145-1 a COD17/145-4, come puoi vedere il listato del programma esce con il set selezionato dall'istruzione OPEN, indipendentemente dall'uso del codice 17 nell'ultima PRINT eseguita.

CHR$(18) e CHR$(146) MODI RVS-ON E RVS-OFF

Il codice 18 predispone la stampa in campo inverso: RVS-ON, mentre il codice 146 predispone la stampa normale: RVS-OFF. Il primo ha validita' locale, cioe' resta valido fino al primo carattere RETURN. Per questa ragione, se non si desidera alternare sulla stessa linea i due modi di stampa, non e' necessario usare il codice 146.
Il programma COD18/146-1, che segue, illustra quanto detto.

```
10 CMD10:A$=""
CARATTERE NORMALE
CARATTERE IN CAMPO INVERSO
CAMBIOCAMBIOCAMBIOCAMBIO
```

```
1 REM COD18/146-1
5 A$="CARATTERE NORMALE"
10 B$="CARATTERE IN CAMPO INVERSO"
15 C$="CAMBIO"
20 OPEN4,4:CMD4
25 PRINTA$
30 PRINTCHR$(18)B$CHR$(146)
35 FORK=1TO2
40 PRINTCHR$(18)C$CHR$(146)C$;
45 NEXTK:PRINT
50 LIST:PRINT#4
55 CLOSE4:STOP
```

Il modo RVS-ON puo' essere usato solo per la stampa in modo testo, carattere normale o allargato.

Questi due codici possono essere usati con lo stesso effetto per la stampa sul video. Segue il programma GETPEEK per chiarire ulteriormente questo argomento.

```
PIPPO DIRETTO E INVERSO
VALORI LETTI CON PEEK:
 16  9  16  16  15  144  137  144  144  143
VALORI LETTI CON GET:
 80  73  80  80  79  80  73  80  80  79
CARATTERI IN CAMPO INVERSO:
P I P P O P I P P O
```

```
1 REM GETPEEK
10 PRINT"PIPPOPIPPO"
15 FORK=0TO9:A(K)=PEEK(3072+K):NEXTK
20 PRINT"";:OPEN3,3
25 FORK=0TO9:GET#3,A$(K):NEXTK
30 PRINT#3,"";
35 FORK=0TO9:PRINT#3,A(K);:NEXTK:PRINT#3
40 FORK=0TO9:PRINT#3,ASC(A$(K));:NEXTK
45 PRINT#3:CLOSE3
47 OPEN4,4:PRINT#4,"PIPPO DIRETTO E INVERSO
49 PRINT#4,"VALORI LETTI CON PEEK: "
50 FORK=0TO9:PRINT#4,A(K);:NEXTK:PRINT#4
51 PRINT#4,"VALORI LETTI CON GET: "
55 FORK=0TO9:PRINT#4,ASC(A$(K));:NEXTK:PRINT#4
57 PRINT#4,"CARATTERI IN CAMPO INVERSO: "
58 FORK=0TO9:PRINT#4,CHR$(18)A$(K);" ";
59 NEXTK:PRINT#4
99 CMD4:LIST:PRINT#4:CLOSE4:STOP
```

COMMENTO A GETPEEK

.10: stampa sulla prima riga del video la parola PIPPO, prima in modo normale (campo diretto), e poi in campo inverso, usando i caratteri di controllo all'interno della stringa di stampa.
.15: legge dalla memoria video, che inizia in 3072, i 10 caratteri stampati, usando la funzione PEEK, e li memorizza in A(K).
.20: porta il cursore nella prima posizione del video, e apre il video come file.
.25: legge dal file video con GET# i 10 caratteri presenti e li memorizza in A$(K).
.30: scende con il cursore sul video di 8 righe.
.35: stampa sul video i codici dei 10 caratteri letti con PEEK; essi sono i D/CODE dei caratteri ed e' riconoscibile il codice che determina il campo inverso. Troviamo infatti: 16, 9, 16, 16, 15, per PIPPO in campo diretto e: 144, 137, 144, 144, 143, per PIPPO in campo inverso, cioe' ogni codice e' aumentato di 128.
.40: stampa sul video i codici dei caratteri letti con GET#, usando la funzione ASC. Come puoi vedere questi codici sono tutti uguali; essi infatti sono i codici ASCII dei caratteri e non si ha differenza tra diretto e inverso.
.45: chiude il file video.
.47: apre la stampante e stampa il titolo.
.49/50: stampa i valori letti con PEEK.
.51/55: stampa i valori letti con GET#.
.57: stampa il titolo.
.58/59: stampa i codici ASCII dei caratteri dopo il codice 18, ottenendo il campo inverso.
.99: trasferisce l'uscita video alla stampante, lista se stesso e chiude correttamente.

==

CHR$(26) RIPETIZIONE CARATTERE GRAFICO

Questo codice di controllo deve essere usato dopo essere entrati in modo grafico con il codice 8. Esso deve essere seguito da un numero N, che specifica quante volte deve essere ripetuta la colonna di punti che segue; tale numero puo' variare da 0 a 255 e deve essere passato con la funzione CHR$(N). Se N=0 il carattere viene ripetuto 256 volte. Se vuoi ripetere per un numero maggiore di volte devi usare la sequenza CHR$(26)CHR$(N)...; piu' volte. Dopo CHR$(N) deve

comparire il codice che rappresenta la colonna di punti da ripetere, passato con la funzione CHR$.

Il programma COD26-1 esemplifica l'uso del codice 26 per allargare il nostro omino, gia' usato in qualche esempio, agendo su ogni colonna di punti che lo compone.

```
1 REM COD26-1
5 OPEN10,4
10 CMD10
15 DATA0,65,67,102,61,31,61,102,67,65,0
20 FORK=1TO5
25 FORI=1TO11
30 READA:A$=CHR$(A+128)
35 PRINTCHR$(8)CHR$(26)CHR$(2)A$;
40 NEXTI
45 RESTORE:NEXTK
50 PRINT:PRINT
55 FORK=1TO5
60 FORI=1TO11
65 READA:A$=CHR$(A+128)
70 PRINTCHR$(8)CHR$(26)CHR$(3)A$;
75 NEXTI
80 RESTORE:NEXTK
85 PRINT:PRINTCHR$(15)
90 PRINT#10:CLOSE10
```

RISULTATI COD26-1

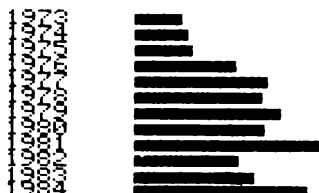
COMMENTO A COD26-1

.5/10: apre la stampante con lfn=10 e trasferisce l'uscita video al file logico 10.
.15: linea DATA che contiene la codifica dell'omino, senza l'aggiunta di 128 ad ogni codice.
.20/45: ripete ciclicamente 5 volte la stampa dell'omino. Legge un codice per volta, aggiunge 128, trasforma in stringa in A$, stampa 2 volte ogni colonna di punti con la sequenza CHR$(8)CHR$(26)CHR$(2)A$. Il

codice 8 poteva essere usato una sola volta fuori dal ciclo. RESTORE rende di nuovo attiva la linea DATA.
.50/75: esegue di nuovo la stampa allargata dell'omino ripetendo 3 volte ogni colonna di punti.
.85/90: ripristina il modo carattere e chiude correttamente.

Il codice 26 e' molto utile per stampare istogrammi orizzontali. Nel programma COD26-2 abbiamo realizzato un esempio.

```
1 REM COD26-2
3 SA=7
5 RESTORE:OPEN10,4,SA
10 CMD10:G$=CHR$(252)
15 DATA18,20,22,38,50,48,55,49,70,39,45,65
20 PRINTCHR$(14)"ANDAMENTO VENDITE"
23 PRINT"ANNI 1973/1984"
25 PRINT
30 FORI=1TO12
35 READB
40 C=1972+I
45 PRINTCHR$(15)C;"    ";CHR$(8)CHR$(26)CHR$(B)G$
50 NEXTI
51 PRINT:PRINT
53 IFSA=7THENSA=0:PRINT#10:CLOSE10:GOTO5
55 PRINT#10,CHR$(15):CLOSE10:STOP
```

RISULTATI COD26-2

COMMENTO A COD26-2

.3: pone SA=7, per attivare il set minuscolo/maiuscolo.

.5/10: esegue il RESTORE e apre la stampante. Trasferisce l'uscita video alla stampante e definisce il carattere grafico G$, che corrisponde a una colonna di punti con spenti i due punti piu' in alto; cosi' le barrette dell'istogramma non si toccano.

.15: linea DATA con 12 numeri che rappresentano l'andamento delle vendite in 12 anni.

.20/25: stampa l'intestazione della tabella.

.30/51: stampa le 12 linee della tabella. Per ogni linea stampa in modo carattere l'anno e in modo grafico la barretta ripetendo il carattere G$ tante volte quanto e' il valore B. Nota che va a capo in modo grafico, e quindi i numeri degli anni risultano un po' ravvicinati verticalmente.

.53: se SA=7 pone SA=0 e ritorna alla linea 5 dopo aver chiuso il file. Cosi' stampa una seconda volta la tabella, ma con l'intestazione nell'altro set di caratteri.
.55: se SA=0 ritorna in modo carattere e chiude correttamente.

CHR$(27) SPOSTAMENTO POSIZIONE GRAFICA

Consente di spostare la posizione di inizio della stampa in uno dei 480 punti di una linea, da 0 a 479. Esso puo' essere usato sia in modo testo che in modo grafico. Non puo' pero' essere usato da solo, ad esso deve seguire il codice 16, seguito a sua volta dalla posizione del punto espressa in due byte, HI e LO, passati con la funzione CHR$. Per esempio se vuoi stampare a partire dal punto 323, devi eseguire il seguente calcolo:

```
X=INT(323/256)
Y=323-X*256
```

e usare nell'istruzione PRINT la sequenza:

```
CHR$(27)CHR$(16)CHR$(X)CHR$(Y)...
```

Nel programma COD27-1 stampiamo un gruppo di omini sovrapposti iniziando la prima volta nel punto 33, la seconda nel punto 22, la terza nel punto 11 e la quarta nel punto 0. I due byte HI e LO che danno la posizioni di inizio devono essere sempre passati, anche se il byte HI e' nullo.

```
1 REM COD27-1
5 OPEN10,4,7:CMD10
10 DATA0,65,67,102,61,31,61,102,67,65,0
15 A$(I)="":FORJ=1TO11:READA:A=A+128
20 A$=A$+CHR$(A):NEXTJ
25 FORK=0TO4
30 PRINTCHR$(8)CHR$(27)CHR$(16)CHR$(0);
35 PRINTCHR$(11*(3+K))A$;
40 NEXTK:PRINT
45 FORK=0TO6
50 PRINTCHR$(8)CHR$(27)CHR$(16)CHR$(0);
55 PRINTCHR$(11*(2+K))A$;
```

```
60 NEXTK:PRINT
65 FORK=0TO8
70 PRINTCHR$(8)CHR$(27)CHR$(16)CHR$(0);
75 PRINTCHR$(11*(1+K))A$;
80 NEXTK:PRINT
85 FORK=0TO10
90 PRINTCHR$(8)A$;
95 NEXTK:PRINT
100 PRINT:PRINTCHR$(15)
105 PRINT#10:CLOSE10
```

RISULTATI COD27-1

COMMENTO A COD27-1

.5: apre il file della stampante con lfn=10 e trasferisce l'uscita video ad esso.
.10: linea DATA che definisce l'omino.
.15/20: costruzione in A$ dell'omino.
.25/40: stampa ciclica di 5 omini a partire dalle colonne: 33, 44, 55, 66, 77.
.45/60: stampa ciclica di 7 omini a partire dalle colonne: 22, 33, 44, 55, 66, 77, 88.
.65/80: stampa ciclica di 9 omini a partire dalle colonne: 11, 22, 33, 44, 55, 66, 77, 88, 99.
.85/95: stampa ciclica di 11 omini a partire dalla colonna 0.
.100/105: ritorno alla stampa in modo carattere e chiusura corretta.
Nella stampa delle prime 3 linee bastava posizionarsi al primo punto, dopo essere passati in grafica, fuori ciclo e poi proseguire la stampa in ciclo senza ulteriori posizionamenti.

Nel programma COD27-2 stampiamo 7 tacche nelle posizioni punto 0, 50, 100, 150, 200, 250 e 300. Abbiamo stampato una linea di numeri per rendere riconoscibili le posizioni delle tacche.

```
1 REM COD27-2
3 OPEN10,4
5 CMD10
7 FORK=0TO4:FORI=0TO9:PRINTCHR$(48+I);
9 NEXTI:NEXTK:PRINT
11 FORI=0TO6
13 A=50*I
15 B=INT(A/256)
17 C=A-B*256
19 PRINTCHR$(8)CHR$(27)CHR$(16);
21 PRINTCHR$(B)CHR$(C)CHR$(255);
23 NEXTI:PRINT#10,CHR$(15):CLOSE10
```

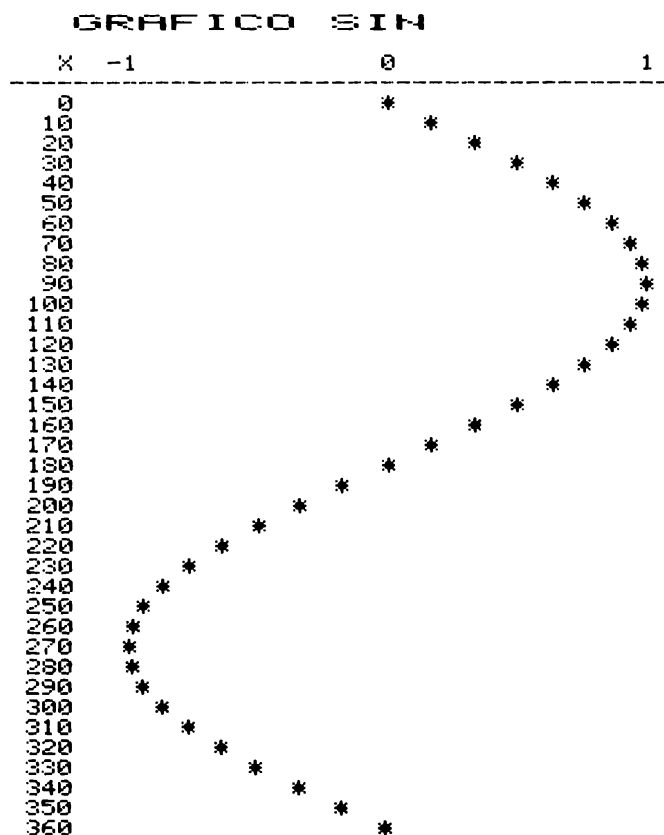
RISULTATI COD27-2

```
01234567890123456789012345678901234567890123456789
|        |        |        |        |        |        |
```

Nota come viene calcolata la posizione punto alle linee 13/17.

Come ultimi esempi dell'uso dei codici 27 e 16 riportiamo i programmi GRAFICO1 e GRAFICO2.

```
1 REM GRAFICO1
3 OPEN4,4:CMD4
5 D$=CHR$(14):N$=CHR$(15)
7 P$=CHR$(16):G$=CHR$(27)
9 C=23:A=16:O=4
11 A$="-":FORI=0TOC+A:A$=A$+"-":NEXT
13 S$="        "
15 PRINTD$"  GRAFICO SIN"
17 PRINTN$
19 PRINTLEFT$(S$,O-1)+"X";
21 PRINTSPC(C-A-O-1)"-1";
23 PRINTSPC(A-1)"0";
25 PRINTSPC(A-1)"1"
27 PRINTA$
29 FORI=0TO360STEP10
31 I$=RIGHT$(S$+STR$(I),O)
33 YO=C*6+A*6*SIN(I*π/180)
35 YH=INT(YO/256):YL=YO-YH*256
37 PRINTI$G$P$CHR$(YH)CHR$(YL)"*"
39 NEXTI:PRINT#4,N$:CLOSE4:STOP
```

RISULTATO GRAFICO1

COMMENTO A GRAFICO1

.3: apre la stampante e trasferisce ad essa l'uscita video.

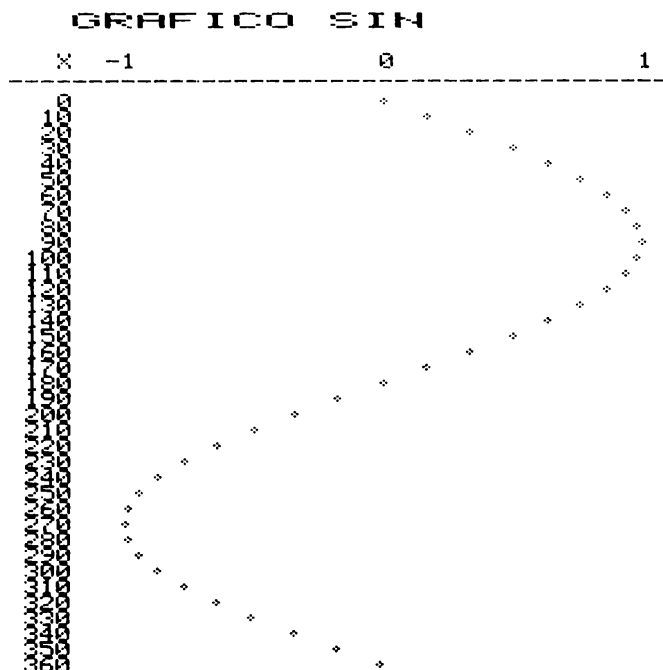
.5/7: definisce come stringhe i caratteri di controllo 14, 15, 16 e 27.

.9: inizializza alcune costanti.
.11/13: prepara le stringhe A$ e S$.
.15: stampa l'intestazione in caratteri allargati.
.17/25: stampa X, -1, 0, 1 usando la funzione SPC per posizionarsi.
.27: stampa le lineette.
.29/39: ciclo di stampa della funzione SIN tra 0 e 360, calcolando le coordinate dei punti e stampandoli con il carattere asterisco in modo testo. Il posizionamento avviene con CHR$(27)CHR$(16) seguiti dalla posizione punto calcolata alle linee 33/35.
Questo grafico e' ottenuto lavorando in modo testo.

Il programma GRAFICO2, invece, lavora in modo grafico, stampando un piccolo rombo definito in F$ con 3 colonne di punti. Come puoi vedere dai risultati il grafico risulta molto piu' compatto.

```
1 REM GRAFICO2
3 OPEN4,4:CMD4
5 D$=CHR$(14):N$=CHR$(15):GR$=CHR$(8)
7 P$=CHR$(16):G$=CHR$(27):NO$=CHR$(15)
9 C=23:A=16:O=4:F$=CHR$(136)+CHR$(148)+CHR$(136)
11 A$="-":FORI=0TOC+A:A$=A$+"-":NEXT
13 S$="          "
15 PRINTD$"  GRAFICO SIN"
17 PRINTN$
19 PRINTLEFT$(S$,O-1)+"X";
21 PRINTSPC(C-A-O-1)"-1";
23 PRINTSPC(A-1)"0";
25 PRINTSPC(A-1)"1"
27 PRINTA$
29 FORI=0TO360STEP10
31 I$=RIGHT$(S$+STR$(I),O)
33 YO=C*6+A*6*SIN(I*π/180)
35 YH=INT(YO/256):YL=YO-YH*256
37 PRINTNO$I$GR$G$P$CHR$(YH)CHR$(YL)F$
39 NEXTI:PRINT#4,N$:CLOSE4:STOP
```

RISULTATI GRAFICO2

```
               GRAFICO SIN
 X  -1                   0                   1
----------------------------------------------
  0                      *
 10                         *
 20                             *
 30                                *
 40                                   *
 50                                     *
 60                                       *
 70                                         *
 80                                          *
 90                                          *
100                                          *
110                                         *
120                                       *
130                                     *
140                                   *
150                                *
160                             *
170                         *
180                      *
190                   *
200               *
210            *
220         *
230       *
240     *
250   *
260  *
270  *
280  *
290   *
300     *
310       *
320         *
330            *
340               *
350                   *
360                      *
```

CHR$(145)   PASSAGGIO SET MAIUSCOLO/GRAFICO

Questo codice fa passare al set maiuscolo/grafico in modo temporaneo, cioe' con validita' limitata, fino al primo carattere RETURN. Rimane preponderante la definizione del set di caratteri operata con il valore di "sa" al momento della OPEN.
Seguono 5 programmi esempio, nei quali si usa il codice 145 e il codice 17 in diverse combinazioni con il valore di SA. Ogni programma e' preceduto dai suoi risultati, dato che termina listando se stesso.

```
commodore
COMMODORE


1 rem cod17/145-1
2 open10,4,7
3 cmd10
4 printchr$(17)"commodore"
5 printchr$(145)"commodore"
6 printchr$(17):print
7 list:print#10:close10
```

```
commodore
COMMODORE

1 rem cod17/145-2
2 open10,4,7
3 cmd10
4 printchr$(17)"commodore"
5 printchr$(145)"commodore"
6 list:print#10:close10
```

```
commodore
COMMODORE


1 REM COD17/145-3
2 OPEN10,4
3 CMD10
4 PRINTCHR$(17)"COMMODORE"
5 PRINTCHR$(145)"COMMODORE"
6 PRINTCHR$(17):PRINT
7 LIST:PRINT#10:CLOSE10
```

```
commodore
COMMODORE

1 REM COD17/145-4
2 OPEN10,4
3 CMD10
4 PRINTCHR$(17)"COMMODORE"
5 PRINTCHR$(145)"COMMODORE"
6 LIST:PRINT#10:CLOSE10
```

```
ABCDEFGHIabcdefghi
AaBbCcDdEeFfGgHh
♠A|B¬C¬D¬E─F|G|H

1 REM COD17/145-5
2 S1$=CHR$(145):S2$=CHR$(17)
3 OPEN10,4:CMD10
4 PRINTS1$"ABCDEFGHI";
5 PRINTS2$"ABCDEFGHI"
6 FORK=0TO7
7 PRINTS1$CHR$(65+K)S2$CHR$(65+K);
8 NEXTK:PRINT
9 FORK=0TO7
10 PRINTS1$CHR$(97+K)S2$CHR$(97+K);
11 NEXTK:PRINT
12 LIST:PRINT#10:CLOSE10
```

2.4 GESTIONE DEL BUFFER DI STAMPA

Il buffer della MPS-803 puo' contenere 90 caratteri; si ha la stampa automatica quando il buffer e' pieno. La stampa puo' aver luogo producendo lo svuotamento totale o parziale del buffer. Nel buffer vengono memorizzati tutti i caratteri che il calcolatore invia, quelli stampabili, i caratteri separatori, le funzioni di stampa, i caratteri di controllo.
Quando usi un programma che manda dati alla stampante, non sempre vedrai uscire dati in corrispondenza all'esecuzione di istruzioni PRINT; se non operi bene puoi chiudere il file di stampa con CLOSE senza aver svuotato completamente il buffer (questo non succede se prima della CLOSE usi un'istruzione PRINT senza lista dati).
Riassumiamo le condizioni nelle quali avviene la stampa:

.1) Il buffer e' pieno, cioe' contiene 90 caratteri, ma i caratteri di testo stampabili sono meno di 80, o tra caratteri di testo e caratteri grafici sono presenti meno di 480 punti. Il buffer viene svuotato completamente producendo una linea di stampa senza andare a capo (infatti non e' stato incontrato un carattere di codice 10 o 13 che manda a capo).

.2) Il buffer contiene meno di 90 caratteri, ma riceve un carattere di "vai a capo"; si ha la stampa di tutti i caratteri con svuotamento completo del buffer e si va a capo.

.3) Il buffer non e' pieno, ma contiene tra caratteri di testo e grafici piu' di 480 punti stampabili; si ha la stampa di una linea con vai a capo, e il buffer viene svuotato dei caratteri stampati.

Il programma ST-AUTOM esemplifica quanto detto.

```
1 REM ST-AUTOM
2 M0$="STAMPA 80 CARATTERI, CHR$(13) FINALE"
3 M1$="STAMPA 90 CARATTERI, CHR$(13) FINALE"
4 M2$="STAMPA 90 CARATTERI, CON ; FINALE"
5 M3$="STAMPA 8 VOLTE CHR$(10); E POI 70"
6 M3$=M3$+" CARATTERI, CON CHR$(13); FINALE"
7 LF$=CHR$(10):CR$=CHR$(13)
8 A$="0123456789":B$=""
9 A1$="ABCDEFGHIJKLMNOPQRSTUVWXYZ":B1$=""
10 FORK=1TO8:B$=B$+A$:NEXTK
11 FORK=1TO4:B1$=B1$+A1$:NEXTK
12 M4$="STAMPA 12 VOLTE CHR$(13); POI 26 "
13 M5$=LEFT$(M4$,26)
14 M4$=M4$+"CARATTERI CON CHR$(13) FINALE"
15 M5$=M5$+"POI LE 10 CIFRE CON ; FINALE"
101 OPEN4,4
103 PRINT#4,M0$
105 PRINT#4,B$:GOSUB200
107 PRINT#4,M1$
109 PRINT#4,B$;A$:GOSUB200
111 PRINT#4,M2$
113 PRINT#4,B$;A$;:GOSUB200
115 PRINT#4,M3$
117 PRINT#4,LF$;LF$;LF$;LF$;LF$;LF$;LF$;
119 PRINT#4,LF$;LEFT$(B$,70);CR$;
121 PRINT#4,M4$
123 PRINT#4,CR$;CR$;CR$;CR$;CR$;CR$;CR$;
125 PRINT#4,CR$;CR$;CR$;CR$;CR$;A1$:GOSUB200
127 PRINT#4,M3$
129 PRINT#4,LF$;LF$;LF$;LF$;LF$;LF$;LF$;
131 PRINT#4,LF$;LEFT$(B1$,70);CR$;
133 PRINT#4,M5$
135 PRINT#4,"LE 10 CIFRE RESTANO NEL BUFFER"
137 PRINT#4,CR$;CR$;CR$;CR$;CR$;CR$;CR$;
139 PRINT#4,CR$;CR$;CR$;CR$;CR$;A$;:GOSUB200
141 CLOSE4:STOP
200 FORI=1TO1000:NEXTI:RETURN
```

Non riportiamo i risultati del programma, che potrai ottenere tu eseguendolo. Esso prepara diverse stringhe di diversa lunghezza e le stampa; dopo ogni stampa viene chiamata una routine che provoca un ciclo di

attesa e che ti permette di vedere cosa e' stato stampato fino a quel momento. Ricorda che se la lista di stampa termina senza punteggiatura il sistema aggiunge un CHR$(13) e quindi si ha la stampa con a capo. Se la lista di stampa termina con ";" non viene aggiunto alcun carattere, mentre se essa termina con "," si ha l'aggiunta di 10 spazi.
Nel programma ST-AUTOM, dato che l'ultima lista di stampa termina con ";" (linea 139) e alla linea 141 si ha la CLOSE del file, gli ultimi caratteri restano nel buffer e non sono stampati. Prova alla fine del programma ad eseguire in immediato:

```
OPEN4,4:PRINT#4:CLOSE4
```

e li vedrai uscire.

2.5 COME SI COMPONE UNO STAMPATO

Nella preparazione degli stampati si presentano due problemi: gli allineamenti orizzontali e le spaziature verticali. Per quanto riguarda il primo problema abbiamo gia' visto in alcuni esempi che si puo' agire sulla posizione di stampa con:

- .le funzioni TAB e SPC,
- .la punteggiatura "," e ";",
- .i codici di controllo 16 e 27 seguiti da opportuni parametri,
- .le funzioni LEN, LEFT$, MID$ e RIGHT$, che consentono di preparare dati stringa tutti della stessa lunghezza.

In generale si allineano i numeri a destra o al punto decimale, e le parole a sinistra.
Per quanto riguarda il problema delle spaziature verticali, dato che la nostra stampante non contiene un contarighe automatico (con relativi codici di controllo per usufruirne), dobbiamo creare noi una routine che conti le righe di avanzamento della carta e ci consenta di intervenire per andare a nuovo foglio o per posizionarci a una determinata riga.
E' necessario conoscere il numero di righe del modulo che si usa e stabilire di quante righe deve essere il margine non stampato.
Il sottoprogramma CONTARIGHE ci consente di stampare su un foglio lungo 66 righe, con un margine di 6 righe. Se scriviamo GOSUB7000 otteniamo di contare una riga di stampa, andando a nuovo foglio se ne sono gia' state scritte 60. Se scriviamo GOSUB7040 otteniamo di andare a nuovo foglio comunque.

```
7000 REM CONTARIGHE
7005 IFCR<>0THEN7020
7010 NR=66:REM LUNGHEZZA FOGLIO
7015 RM=6:REM RIGHE DI MARGINE
7020 IFCR=NR-RMTHEN7030:REM CAMBIO FOGLIO
7025 CR=CR+1:RETURN:REM +1 IN CONTA RIGHE
7030 FORR=1TORM:PRINT#4:NEXTR:REM CAMBIA FOGLIO
7035 CR=1:RETURN:REM RICOMINCIA FOGLIO
7040 REM ENTRATA PER CAMBIARE FOGLIO
7045 FORR=CR+1TOMR+NR:PRINT#4:NEXTR:GOTO7035
```

Quando si chiama la prima volta il sottoprogramma con GOSUB7000 deve essere CR=0, e allora viene inizializzato NR=66 e RM=6 (puoi cambiare queste costanti secondo le tue esigenze). Quando, invece CR<>0, la routine va a nuovo foglio se necessario e incrementa il contatore di riga. L'entrata 7040 fa andare a nuovo foglio indipendentemente dal numero di righe gia' stampate.

Seguono alcuni esempi relativi agli allineamenti orizzontali; essi stampano i risultati e poi listano se stessi (puoi evitare la lista cancellando LIST).

Il programma INC1 esemplifica l'uso della funzione TAB, dopo aver trasferito la stampa dal video alla stampante con CMD.

```
0123456789012345678901234567890123456789
POS0      POS10               POS30

NUOVA RIGA


1 REM INC1
3 OPEN4,4:CMD4
5 A$="0123456789":B$=""
7 FORK=1TO4:B$=B$+A$:NEXTK
9 PRINTB$
11 PRINTTAB(0)"POS0";TAB(6)"POS10";
13 PRINTTAB(15)"POS30"
15 PRINTTAB(80);"NUOVA RIGA"
17 PRINT:LIST
19 PRINT#4:CLOSE4:STOP
```

Il programma INC2 esemplifica l'uso della funzione SPC, dopo aver trasferito la stampa dal video alla stampante con CMD.

```
0123456789012345678901234567890123456789
DIECI CAR.          DIECI CAR.
DIECI CAR.          DIECI CAR.
```

```
1 REM INC2
3 OPEN4,4:CMD4
5 A$="0123456789":B$=""
7 FORK=1TO4:B$=B$+A$:NEXTK
9 PRINTB$
11 C$="DIECI CAR."
13 PRINTC$;SPC(10);C$
15 PRINTC$TAB(10)C$
17 PRINT:LIST
19 PRINT#4:CLOSE4:STOP
```

Puoi modificare INC1 e INC2 per stampare con PRINT#, invece che con CMD; ottieni gli stessi risultati.

Il programma INC5 esemplifica l'uso della punteggiatura finale nella lista di stampa. Esso stampa con CMD.

```
0123456789012345678901234567890123456789
ABC           DEF
PRIMA PAROLA            SECONDA PAROLA
ABCDEF
PRIMA PAROLASECONDA PAROLA
```

```
1 REM INC5
3 OPEN4,4:CMD4
5 A$="0123456789":B$=""
7 FORK=1TO4:B$=B$+A$:NEXTK
9 PRINTB$
11 PRINT"ABC","DEF"
13 PRINT"PRIMA PAROLA","SECONDA PAROLA"
15 PRINT"ABC";"DEF"
17 PRINT"PRIMA PAROLA";"SECONDA PAROLA"
19 PRINT:LIST
21 PRINT#4:CLOSE4:STOP
```

Puoi modificare INC5, stampando con PRINT#, invece che con CMD e ottenendo gli stessi risultati.

Il programma INC7 mostra come incolonnare i numeri riducendoli tutti della stessa lunghezza, dopo averli

trasformati in stringa, con l'aggiunta di spazi a sinistra, usando la funzione RIGHT$.

```
0123456789012345678901234567890123456789
 1234     13567     45     890
 5432     9876     3456     12345

TABELLA NUMERI IN COLONNA

0123456789012345678901234567890123456789
      1234      13567         45       890
      5432       9876       3456     12345
```

```
1 REM INC7
5 DIMC(7),C$(7):SP$="          "
10 OPEN4,4:CMD4
15 A$="0123456789":B$=""
20 FORK=1TO4:B$=B$+A$:NEXTK
25 PRINTB$
30 DATA1234,13567,45,890
35 DATA5432,9876,3456,12345
40 FORK=0TO7:READC(K)
45 C$(K)=STR$(C(K)):NEXTK
50 FORJ=0TO1:FORK=0TO3:PRINTC(K+J*4);"   ";
55 NEXTK:PRINT:NEXTJ
60 T$="TABELLA NUMERI IN COLONNA"
65 PRINT:PRINTT$:PRINT
70 PRINTB$
75 FORJ=0TO1:FORK=0TO3
80 PRINTRIGHT$(SP$+C$(K+J*4),10);
85 NEXTK:PRINT:NEXTJ
90 PRINT#4:CMD4:LIST:PRINT#4:CLOSE4:STOP
```

Il programma INC8 mostra come incolonnare i numeri ricorrendo, dopo la trasformazione in stringa, alle funzioni SPC e LEN.

```
0123456789012345678901234567890123456789
 1234     8976     3456     13567
 45     890     5432     9876

TABELLA NUMERI IN COLONNA

0123456789012345678901234567890123456789
      1234       8976       3456     13567
        45        890       5432      9876
```

```
1 REM INC8
5 DIMC(7),C$(7):SP$="           "
10 OPEN4,4:CMD4
15 A$="0123456789":B$=""
20 FORK=1TO4:B$=B$+A$:NEXTK
25 PRINTB$
30 DATA1234,8976,3456,13567,45,890,5432,9876
35 FORK=0TO7:READC(K):C$(K)=STR$(C(K)):NEXTK
40 FORJ=0TO1:FORK=0TO3:PRINTC(K+J*4);"   ";
45 NEXTK:PRINT:NEXTJ
50 T$="TABELLA NUMERI IN COLONNA"
55 PRINT:PRINTT$:PRINT:PRINTB$
60 FORJ=0TO1:FORK=0TO3
65 PRINTSPC(10-LEN(C$(K+J*4)))C$(K+J*4);
70 NEXTK:PRINT:NEXTJ
75 PRINT#4:CMD4:LIST:PRINT#4:CLOSE4:STOP
```

Il programma INC9 mostra come allineare parole ricorrendo alle funzioni SPC e LEN.

```
BELLO   BENISSIMO   ALLEGRO   PIACEVOLE
RILASSANTE   SOLEGGIATO   GRADEVOLE   UTILE
DEFINITIVO   ARIOSO   STUPENDO   MAGNIFICO

BELLO         BENISSIMO     ALLEGRO       PIACEVOLE
RILASSANTE    SOLEGGIATO    GRADEVOLE     UTILE
DEFINITIVO    ARIOSO        STUPENDO      MAGNIFICO
```

```
1 REM INC9
5 DIMC$(11)
10 OPEN4,4:CMD4
15 DATABELLO,BENISSIMO,ALLEGRO
20 DATAPIACEVOLE,RILASSANTE,SOLEGGIATO
25 DATAGRADEVOLE,UTILE,DEFINITIVO
30 DATAARIOSO,STUPENDO,MAGNIFICO
35 FORK=0TO11:READC$(K):NEXTK
40 FORJ=0TO2:FORK=0TO3:PRINTC$(K+J*4);"   ";
45 NEXTK:PRINT:NEXTJ
50 N=0:FORK=0TO11
55 IFLEN(C$(K))>NTHENN=LEN(C$(K))
60 NEXTK:PRINT:PRINT
65 N=N+3:FORJ=0TO2:FORK=0TO3
70 PRINTC$(K+J*4);SPC(N-LEN(C$(K+J*4)));
75 NEXTK:PRINT:NEXTJ
80 PRINT#4:CMD4:LIST:PRINT#4:CLOSE4:STOP
```

2.6 COPIA DEL VIDEO TESTO SU CARTA

In generale con il termine "copia del video su carta" (hardcopy) si intende uno stampato che corrisponda al contenuto del video. Tale stampato puo', nel nostro caso:

.Contenere gli stessi caratteri che compaiono sul video, grafici o di testo, ma non essere identico al video. Infatti sul video i caratteri sono rappresentati in una matrice 8x8, mentre sulla carta sono rappresentati in una matrice 6x7.

.Contenere una immagine identica ottenuta riproducendo esattamente i punti del video (con dimensioni diverse naturalmente).

Noi in questo paragrafo ci occupiamo della copia del video su carta, quando il video e' in modo testo; nel prossimo paragrafo trattiamo invece l'argomento del video grafico.
In modo testo la mappa del video occupa 1000 byte; l'indirizzo del primo byte si puo' ottenere moltiplicando per 256 il contenuto del byte 1342. Al momento dell'accensione la mappa video inizia al byte 3072 (da 3072 a 4071) e la mappa degli attributi dei caratteri (colore, luminosita', flash) va da 2048 a 3047. Nella mappa video sono contenuti i D/CODE (vedi Paragrafo 4.8) dei caratteri, che sono diversi per i caratteri in campo diretto e in campo inverso.
Abbiamo preparato il sottoprogramma HARD1, che legge con la funzione PEEK i codici D/CODE dei caratteri presenti sul video, li trasforma in codici ASCII, se i D/CODE sono maggiori di 128 (campo inverso) predispone il carattere di controllo per attivare sulla stampante il campo inverso, e li stampa, riproducendo i caratteri del video con i caratteri della stampante. HARD1 puo' quindi servire per copiare su carta un video in modo testo, rispettando il campo diretto e il campo inverso.

Nel programma COPIAVIDEO1 abbiamo usato HARD1 per ricopiare il video due volte, prima con il set maiuscolo/grafico e poi con il set minuscolo/maiuscolo. Prima di far girare il programma, lo abbiamo listato con LIST sul video. Dato che il programma non e' molto lungo, esso entra tutto in un quadro video. Come puoi vedere dal risultato ogni linea e' di 40 caratteri; avresti ottenuto un risultato diverso trasferendo la lista alla stampante con CMD.

```
1 REM COPIAVIDEO1
2 REM COPIA SET MAIUSCOLO/GRAFICO
3 H1$=CHR$(145):GOSUB10000
4 H1$=CHR$(17):GOSUB10000:STOP
10000 REM HARD1
10001 OPEN4,4:PRINT#4
10002 H1=256*PEEK(1342)-40
10003 FORH0=0TO24:H0$=H1$:H1=H1+40
10004 FORH2=H1TOH1+39:H3=PEEK(H2)
10005 IFH3>128THENH3=H3-128:H4=1:H0$=H0$
+CHR$(18)
10006 IF(H3>0)*(H3<32)THENH3=H3+64:GOTO1
0010
10007 IF(H3>31)*(H3<64)THEN10010
10008 IF(H3>63)*(H3<96)THENH3=H3+128:GOT
O10010
10009 IF(H3>95)*(H3<128)THENH3=H3+64:GOT
O10010
10010 H0$=H0$+CHR$(H3)
10011 IFH4=1THENH0$=H0$+CHR$(146):H4=0
10012 NEXTH2:PRINT#4,H0$:NEXTH0
10013 PRINT#4:CLOSE4:RETURN

RUN
```

```
1 rem copiavideo1
2 rem copia set maiuscolo/grafico
3 h1$=chr$(145):gosub10000
4 h1$=chr$(17):gosub10000:stop
10000 rem hard1
10001 open4,4:print#4
10002 h1=256*peek(1342)-40
10003 forh0=0to24:h0$=h1$:h1=h1+40
10004 forh2=h1toh1+39:h3=peek(h2)
10005 ifh3>128thenh3=h3-128:h4=1:h0$=h0$
+chr$(18)
10006 if(h3>0)*(h3<32)thenh3=h3+64:goto1
0010
10007 if(h3>31)*(h3<64)then10010
10008 if(h3>63)*(h3<96)thenh3=h3+128:got
o10010
10009 if(h3>95)*(h3<128)thenh3=h3+64:got
o10010
10010 h0$=h0$+chr$(h3)
10011 ifh4=1thenh0$=h0$+chr$(146):h4=0
10012 nexth2:print#4,h0$:nexth0
10013 print#4:close4:return

run
```

COMMENTO A COPIAVIDEO1

.3: pone H1$=CHR$(145) per attivare il set maiuscolo/grafico e chiama il sottoprogramma HARD1.
.4: pone H1$=CHR$(17) per attivare il set minuscolo/maiuscolo e chiama il sottoprogramma HARD1; poi si ferma.

COMMENTO A HARD1

.10001: apre la stampante.
.10002: prepara in H1 l'indirizzo di inizio della mappa video - 40.
.10003: inizia un ciclo per H0 da 0 a 24 per leggere le 25 linee del video. Pone H0$=H1$ e incrementa H1 di 40, per puntare a inizio riga; alla fine della lettura di ogni riga torna ad eseguire queste due ultime operazioni.
.10004: inizia il ciclo per leggere i 40 caratteri di una riga. Legge il D/CODE di un carattere con la funzione PEEK.
.10005: se il D/CODE supera 128, aggiunge alla stringa H0$ il codice 18 per ottenere RVS-ON sulla stampante e pone H4=1.
.10006/10009: trasforma il D/CODE in codice ASCII.
.10010: aggiunge alla stringa H0$ il nuovo carattere.
.10011: se H4=1 aggiunge alla stringa H0$ il codice 146 per tornare a RVS-OFF e pone H4=0.
.10012: torna alla lettura di un nuovo carattere fino alla conclusione del ciclo di carattere nella riga. Poi stampa la stringa H0$ sulla stampante e passa alla prossima riga video fino alla conclusione del ciclo di riga.
.10013: chiude la stampante e ritorna al programma principale.

Il programma COPIAVIDEO2 stampa sul video 3 stringhe in campo diretto e 3 in campo inverso, nel set maiuscolo/grafico, poi chiama HARD1 per ricopiarle sulla carta. Dopo cambia il set di caratteri, stampa le stesse stringhe e chiama HARD1 per eseguire la copia del video.

```
1 REM COPIAVIDEO2
2 PRINT"";:PRINT"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
3 PRINT" H┴┬‾‾▂▂▁▒ ┌ ▄||▌▌ ▛└ ▀▗||"
4 PRINT"●○‾─| |╭┌┘╋|♠♥‾─| |╰└─◆♣─×|╱╲"
5 PRINT"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
6 PRINT"H┴┬‾‾▂▂▁▒ ┌ ▄||▌▌ ▛└ ▀▗|"
7 PRINT"●○‾─| |╭┌┘╋|♠♥‾─| |╰└─◆♣─×|╱╲"
10 H1$=CHR$(145):GOSUB10000
11 H1$=CHR$(17):GOSUB10000:STOP
10000 REM HARD1
10001 OPEN4,4:PRINT#4
10002 H1=256*PEEK(1342)-40
10003 FORH0=0TO24:H0$=H1$:H1=H1+40
10004 FORH2=H1TOH1+39:H3=PEEK(H2)
10005 IFH3>128THENH3=H3-128:H4=1:H0$=H0$+CHR$(18)
10006 IF(H3>0)*(H3<32)THENH3=H3+64:GOTO10010
10007 IF(H3>31)*(H3<64)THEN10010
10008 IF(H3>63)*(H3<96)THENH3=H3+128:GOTO10010
10009 IF(H3>95)*(H3<128)THENH3=H3+64:GOTO10010
10010 H0$=H0$+CHR$(H3)
10011 IFH4=1THENH0$=H0$+CHR$(146):H4=0
10012 NEXTH2:PRINT#4,H0$:NEXTH0
10013 PRINT#4:CLOSE4:RETURN
```

RISULTATI COPIAVIDEO2

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
 H┴┬‾‾▂▂▁▒ ┌ ▄||▌▌ ▛└ ▀▗||
●○‾─| |╭┌┘╋|♠♥‾─| |╰└─◆♣─×|╱╲
ABCDEFGHIJKLMNOPQRSTUVWXYZ


abcdefghijklmnopqrstuvwxyz
 H┴┬‾‾▂▂▁▒ ┌ ▄||▌▌ ▒└ ▀▗||
QWERTYUIOP←+|ASDFGHJKL-ZXCVBNM
abcdefghijklmnopqrstuvwxyz

QWERTYUIOP←+|ASDFGHJKL-ZXCVBNM
```

Il programma COPIAVIDEO3 usa il sottoprogramma HARD2 per copiare il video. In HARD2 la copia del video viene fatta aprendo il video come file, leggendo con GET#

i caratteri, riga per riga, e stampandoli. Dato che i caratteri sono letti in codice ASCII, il programma non rispetta il campo inverso, salvo che per i caratteri contenuti tra virgolette.

```
1 REM COPIAVIDEO3
2 SA=0:GOSUB10000
3 SA=7:GOSUB10000
4 STOP
10000 REM HARD2
10005 PRINT"■";
10007 OPEN3,3:OPEN4,4,SA
10010 FORK=0TO24:A$=""
10015 FORJ=0TO39
10020 GET#3,B$:A$=A$+B$
10025 NEXTJ
10030 PRINT#4,A$
10035 NEXTK:PRINT
10040 CLOSE3:CLOSE4
10045 RETURN

RUN
```

```
1 rem copiavideo3
2 sa=0:gosub10000
3 sa=7:gosub10000
4 stop
10000 rem hard2
10005 print"■";
10007 open3,3:open4,4,sa
10010 fork=0to24:a$=""
10015 forj=0to39
10020 get#3,b$:a$=a$+b$
10025 nextj
10030 print#4,a$
10035 nextk:print
10040 close3:close4
10045 return

run
```

Prima di far girare il programma lo abbiamo listato sul video. Esso viene stampato nei due set di caratteri.

Affrontiamo ora il problema di ottenere su carta una

copia identica del video. Il problema e' abbastanza complesso infatti noi disponiamo dei codici dei caratteri, in D/CODE se li leggiamo con la funzione PEEK. Dato che nel COMMODORE 16 la mappa dei caratteri in ROM descrive solo i caratteri in campo diretto, D/CODE da 0 a 127, per i due set, dobbiamo analizzare il codice, e, se maggiore di 128, andare a prendere la descrizione del codice corrispondente al campo diretto e scambiare i bit 0 con bit 1 e viceversa. Le due mappe di descrizioni dei due set di caratteri occupano ciascuna 1024 byte (128*8=1024) e si trovano in ROM da 53248 (D000H) a 54271, e da 54272 (D400H) a 55296. Per poter ricostruire i caratteri per punti sulla stampante occorrono le descrizioni dei caratteri, che pero' non sono accessibili da BASIC in ROM. In conseguenza i programmi BASIC che vogliono accedere alla descrizione dei caratteri, senza usare routine in linguaggio macchina, devono essere lanciati dopo:

.aver abbassato il top della memoria del BASIC per lasciare liberi sopra almeno 2K di memoria RAM,

.aver trasferito con l'istruzione MONITOR i 2K della ROM caratteri in RAM. L'istruzione MONITOR puo' accedere alla ROM senza problemi.

Abbiamo preparato il sottoprogramma HARD3, che lavora cosi':

.Preleva i D/CODE dal video con la funzione PEEK e li memorizza in una matrice X(M,I), dove M rappresenta il numero delle righe video da ricopiare e I il numero dei caratteri per riga.

.Usa una matrice D a due indici, che abbia un numero di righe multiplo di 7, infatti per i caratteri grafici della stampante si devono usare 7 punti incolonnati, e che sia sufficiente a contenere le descrizioni dei caratteri di ogni riga video, ognuno formato da 8 linee. In tale matrice, che ha 40 colonne, vengono memorizzate le descrizioni di ogni carattere, occupando 8 righe. Le ultime righe della matrice rimangono eventualmente vuote, dato che si adatta una struttura multipla di 8 in una multipla di 7.

.Preleva dalla matrice D i byte a gruppi di 7 (sulla stessa colonna), calcola i codici degli 8 caratteri grafici a cui essi danno luogo (un carattere per ogni colonna di bit) e li memorizza in Z(7), poi li trasforma in stringa e li stampa. La prima riga stampata corrisponde quasi a una riga del video, infatti ha lavorato su 7 linee di punti invece che su 8, ma alla fine l'immagine e' formata dallo stesso numero di

punti del video. Noterai una certa differenza con i caratteri stampati di solito dalla stampante.

Noi abbiamo richiamato il sottoprogramma HARD3 dal programma COPIAVIDEO4. Abbiamo dovuto limitare il numero di righe video da ricopiare, infatti le matrici occupano molto spazio e per poter avere in RAM la descrizione dei caratteri abbiamo dovuto abbassare il top della memoria. Ci siamo limitati a scrivere sul video i caratteri di D/CODE compreso tra 32 e 127, che sono 96 e quindi occupano circa 3 righe video. Abbiamo dimensionato in conseguenza le matrici X e D; X(2,39) per avere tre righe e 40 colonne, e D(27,39) per avere 28 righe e 40 colonne, infatti per 3 righe di caratteri ci vogliono 3x8=24 linee di punti e il multiplo di 7 piu' vicino a 24 e' 28.

```
1 REM COPIAVIDEO4
3 DIMX(2,39),D(27,39),Z(7)
50 H2$=CHR$(142)+CHR$(146)
55 GOSUB100:D=0:GOSUB10000
57 H2$=CHR$(142)+CHR$(18)
59 GOSUB100:D=0:GOSUB10000
70 H2$=CHR$(14)+CHR$(146)
75 GOSUB100:D=1:GOSUB10000
77 H2$=CHR$(14)+CHR$(18)
79 GOSUB100:D=1:GOSUB10000
81 PRINTCHR$(146)CHR$(142):STOP
100 PRINT"J";H2$;
105 FORK=32TO127:PRINTCHR$(K);:NEXTK
115 PRINT:RETURN
10000 REM HARD3
10005 OPEN4,4
10010 H1=256*PEEK(1342)-40
10015 FORH0=0TO2:H1=H1+40
10020 J=0:FORH2=H1TOH1+39
10021 X(H0,J)=PEEK(H2)
10060 J=J+1:NEXTH2:NEXTH0
10063 GOSUB20000
10064 GOSUB20020
10065 PRINT#4:CLOSE4:RETURN
20000 REM PRELEVA DESCRIZIONI
20005 A=14080+D*1024
20007 FORM=0TO2:FORI=0TO39:Y=X(M,I)
20008 SW=0:IFY>127THENSW=1:Y=Y-128
20009 FORK=0TO7:D(M*8+K,I)=PEEK(A+Y*8+K)
20011 IFSW=1THEND(M*8+K,I)=2↑8-1-D(M*8+K,I)
20015 NEXTK:NEXTI:NEXTM:RETURN
20020 PRINT#4,CHR$(8);
20035 FORN=0TO27STEP7
```

```
20038 FORI=0TO39
20039 FORL=0TO7:Z(L)=0:NEXTL
20040 FORM=0TO6:Y=D(M+N,I)
20043 FORL=0TO7
20045 IFINT(Y/2↑(7-L))=0THEN20060
20050 Z(L)=Z(L)+2↑M:Y=Y-2↑(7-L)
20060 NEXTL:NEXTM
20065 A$="":FORL=0TO7:A$=A$+CHR$(Z(L)+128)
20066 NEXTL
20067 PRINT#4,A$;:NEXTI:PRINT#4
20080 NEXTN:PRINT#4,CHR$(15):RETURN
```

RISULTATI COPIAVIDEO4

```
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[£]↑←
```

```
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[£]↑←
```

```
 !"#$%&'()*+,-./0123456789:;<=>?@abcdefg
hijklmnopqrstuvwxyz[£]↑←ABCDEFGHIJKLMNO
PQRSTUVWXYZ
```

```
 !"#$%&'()*+,-./0123456789:;<=>?@abcdefg
hijklmnopqrstuvwxyz[£]↑←ABCDEFGHIJKLMNO
PQRSTUVWXYZ
```

COMMENTO A COPIAVIDEO4

Prima di caricare o scrivere il programma e' necessario eseguire in immediato le seguenti istruzioni:

.1): POKE 55,255:POKE 56,54:CLR per abbassare il top della memoria BASIC a 14079 (36FFH).

.2): MONITOR per attivare il MONITOR

T D000 D7FF 3700 per trasferire i 2K delle descrizioni dei caratteri dalla ROM (53248) in RAM (14080).

X per uscire dal MONITOR.

Segue il commento al programma.

.3: dimensiona le 3 matrici:

X(2,39) per contenere i D/CODE di 3 righe video,

D(27,39) per contenere le descrizioni per punti

delle 3 righe di caratteri, ma con un numero di linee di punti multiplo di 7.

Z(7), per contenere i codici dei caratteri grafici ottenuti da 7 byte incolonnati in D.

.50/55: predispone il set maiuscolo/grafico in campo diretto sul video, chiama il sottoprogramma in 100 per scrivere i caratteri sul video, pone D=0 per puntare al primo set e chiama HARD3 per eseguire la copia.

.57/59: come sopra ma per il campo inverso.

.70/75: come sopra, ma per il secondo set e il campo diretto.

.77/79: come sopra, ma per il campo inverso.

.81: ripristina il set iniziale e si ferma.

Ottieni come risultati 96 caratteri del primo set, prima in campo diretto e poi in campo inverso, e poi 96 caratteri del secondo set, ancora nei due modi.

COMMENTO A HARD3

.10005: apre la stampante.

.10010: prepara l'indirizzo della mappa video.

.10015/10060: trasferisce il contenuto della mappa video nella matrice X, usando la funzione PEEK.

.10063: chiama il sottoprogramma in 20000 per riempire la matrice D con le descrizioni dei caratteri.

.10064 chiama il sottoprogramma in 20020 per stampare la copia del video.

.10065: chiude la stampante e ritorna al programma principale.

.20000/20015: trasferisce le descrizioni dei caratteri nella matrice D; se il D/CODE supera 127 scambia tra loro i bit 0 con bit 1 e viceversa, per ottenere la rappresentazione in campo inverso.

.20020: inizia la parte stampa passando in modo grafico.

.20035: predispone l'analisi della matrice D in strisce di 7 linee.

.20038: predispone l'analisi dei 40 caratteri di una riga.

.20039: azzera il vettore Z(7).

.20040/20060: decodifica a gruppi i 7 byte della striscia, ottenendo i codici di 8 caratteri grafici.

.20065/20066: calcola la stringa corrispondente

.20067: stampa 8 caratteri grafici contenuti in A$ e passa al prossimo gruppo di byte. Alla fine della striscia va a capo.

.20080: chiude il ciclo di analisi delle strisce di D, e alla fine ripassa in modo testo ed esce.

A nostro avviso l'interesse di questo programma sta solo nelle difficolta' di programmazione incontrate, cioe' nell'aver fatto lavorare il BASIC a livello di bit. Infatti il programma risulta piuttosto lento e non puo' essere esteso a tutto il video se non si estende la memoria RAM del calcolatore.
Nel prossimo paragrafo presentiamo un programma in linguaggio macchina che lavora sulla pagina grafica; si potrebbe prepararne uno analogo che lavora sul video in modo testo, riuscendo ad ottenere la copia molto piu' velocemente.

2.7 COPIA DEL VIDEO GRAFICO SU CARTA

La pagina grafica viene attivata passando in modo grafico con l'istruzione GRAPHIC. Essa inizia al byte 8192 ($2000) ed e' formata da 8000 byte, 8 per ogni posizione carattere del video in modo testo. I byte della pagina grafica sono utilizzati come segue:
 .i primi 8 descrivono il primo insieme di 8x8 punti situato nell'angolo in alto a sinistra, corrispondente alla posizione carattere di coordinate 0,0;
 .i successivi 8 byte descrivono l'insieme di 8x8 punti corrispondente alla posizione carattere di coordinate 1,0 (accanto e a destra della precedente), e cosi' via;
 .gli ultimi 8 byte, di indirizzo da 16184 a 16191, descrivono l'ultimo insieme di 8x8 punti corrispondente alla posizione carattere di coordinate 39,24.
Nella Figura 2.4 riportiamo la corrispondenza tra i byte della pagina grafica e le posizioni sul video.
Per risolvere il problema della copia su carta, abbiamo gia' disponibile la descrizione per punti, ma dobbiamo trasformarla in caratteri grafici, cioe' in colonne di 7 punti. Abbiamo gia' eseguito questo lavoro in BASIC nel paragrafo precedente con il sottoprogramma HARD3. In quel caso nella matrice D(27,39) abbiamo messo noi i byte gia' organizzati nella sequenza nella quale danno il disegno del quadro, come appare nella Figura 2.4. In questo caso, invece, la sequenza di byte per indirizzo crescente e' utilizzata in un modo piu' complicato e dobbiamo trovare un opportuno algoritmo che ci permetta di riferirci ad essi. Per questa ragione, dopo attento esame, abbiamo preparato un sottoprogramma in ASSEMBLER, che chiamiamo HARD4; esso viene richiamato dal programma BASIC VIDEOGRAF e si ottiene la copia su carta della pagina grafica.

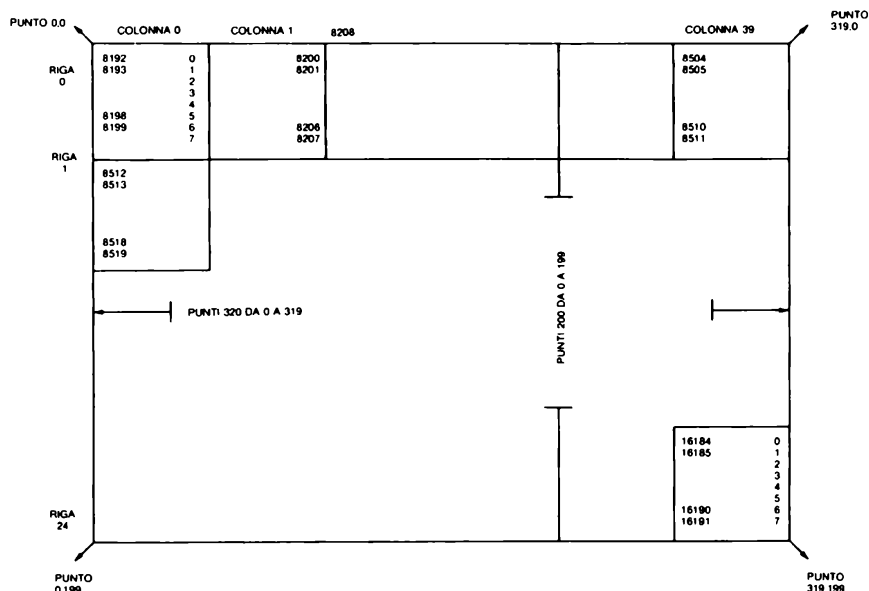



Figura 2.4 Come i byte della pagina grafica danno il quadro video

Vediamo ora quali algoritmi abbiamo usato per prelevare dalla pagina grafica i punti nella sequenza necessaria per la stampa. Dobbiamo costruire caratteri grafici per la stampante, formati da colonne di 7 punti, cioe' utilizzare i byte che danno il quadro video a strisce alte 7 punti. La prima volta, per la prima linea di stampa, da 8192 a 8198, da 8200 a 8206,..., da 8504 a 8510, e cosi' via. Dal momento che 200 diviso 7 da' 28 con resto di 4, l'ultima striscia di caratteri grafici avra' solo 4 punti veri e gli ultimi 3, i piu' significativi a 0.
Consideriamo un singolo punto sul video; le sue coordinate grafiche X e Y danno la posizione, con 0<=X<=319 e 0<=Y<=199. Noi dobbiamo trovare un algoritmo che metta in relazione X e Y con l'indirizzo del byte (da 8192 a 16191) a cui il punto appartiene e con la posizione del bit corrispondente nel byte (da 0, a destra, a 7, a sinistra). Chiamiamo IB l'indirizzo del byte e IP la posizione del bit nel byte, espressa con il suo peso; abbiamo:

```
IB=8192+8*INT(X/8)+320*INT(Y/8)+(Y AND 7)

IP=2^(7 - (X AND 7))
```

Con riferimento alla Figura 2.4, verifichiamo le due formule per il bit di posizione 5 nel byte di indirizzo 8513, e per il bit di posizione 4 nel byte di indirizzo 16184.

.1) byte 8513, bit di posizione 5
coordinata X=2, infatti si trova nella terza colonna da sinistra
coordinata Y=9, infatti si trova nella decima linea di punti dall'alto

```
IB=8192+8*INT(2/8)+320*INT(9/8)+(9 AND 7)
  =8192+0+320*1+1
  =8192+320+1
  =8513
IP=2^(7-(2 AND 7))
  =2^(7-2)
  =2^5
```

.2) byte 16184, bit di posizione 4
coordinata X=315, infatti si trova nella quintultima posizione dell'ultimo byte della riga
coordinata Y=192, infatti si trova nella 193-esima linea di punti dall'alto

```
IB=8192+8*INT(315/8)+320*INT(192/8)+(192 AND 7)
  =8192+8*38+320*24+0
  =8192+312+7680
  =16184
IP=2^(7-(315 AND 7))
  =2^(7-3)
  =2^4
```

```
0 REM VIDEOGRAF
10 POKE56,23:POKE55,0:CLR
20 FORI=0TO244:READA:POKE5888+I,A:NEXT
30 OPEN4,4:PRINT#4,CHR$(8);
40 FORI=0TO321:PRINT#4,CHR$(192);:NEXT
50 CMD4:SYS5888
60 FORI=0TO321:PRINT#4,CHR$(129);:NEXT
61 PRINT#4,CHR$(14)
70 CLOSE4
999 REM DATI PER HARD4
1000 DATA169,7,141,71,63,169,0,141
1010 DATA72,63,169,255,32,210,255,169
1020 DATA0,141,64,63,141,65,63,169
1030 DATA128,141,74,63,169,0,141,73
```

```
1040 DATA63,173,72,63,24,109,73,63
1050 DATA141,68,63,32,121,23,238,73
1060 DATA63,173,73,63,205,71,63,208
1070 DATA232,173,74,63,32,210,255,238
1080 DATA64,63,173,64,63,208,3,238
1090 DATA65,63,201,64,208,201,173,65
1100 DATA63,201,1,208,194,169,255,32
1110 DATA210,255,169,13,32,210,255,173
1120 DATA72,63,24,105,7,141,72,63
1130 DATA201,196,208,5,169,4,141,71
1140 DATA63,173,68,63,201,199,208,146
1150 DATA96,173,64,63,41,248,141,66
1160 DATA63,173,64,63,41,7,141,67
1170 DATA63,173,68,63,74,74,74,141
1180 DATA69,63,173,68,63,41,7,141
1190 DATA70,63,169,32,133,4,169,0
1200 DATA133,3,172,69,63,240,16,165
1210 DATA3,24,105,64,133,3,165,4
1220 DATA105,1,133,4,136,208,240,165
1230 DATA3,24,109,66,63,133,3,165
1240 DATA4,109,65,63,133,4,173,70
1250 DATA63,24,101,3,133,3,165,4
1260 DATA105,0,133,4,169,128,174,67
1270 DATA63,240,4,74,202,208,252,33
1280 DATA3,240,17,169,1,172,73,63
1290 DATA240,4,10,136,208,252,13,74
1300 DATA63,141,74,63,96
```

COMMENTO A VIDEOGRAF

.10: abbassa il top della memoria a $1700 per non sporcare con le variabili il sottoprogramma in linguaggio macchina, le memorie degli attributi per la pagina grafica e la pagina grafica stessa (che occupano rispettivamente i byte da $1700 a $17F4, da $1800 a 1BE7, da 1C00 a 1FE7 e da $2000 a $3F40).

.20: carica il programma in linguaggio macchina in memoria.

.30: apre il canale con la stampante e la pone in modo grafico.

.40: disegna la striscia di contorno superiore.

.50: dirotta l'output sulla stampante e esegue la routine in linguaggio macchina.

.60: disegna la striscia di contorno inferiore e fa uscire la stampante dal modo grafico.

.70: chiude il canale con la stampante.

.1000/1300: dati relativi a HARD4.

Vediamo ora il programma ASSEMBLER HARD4:

```
MONITOR

   PC  SR AC XR YR SP
; 0000 00 00 00 00 F8

. 1700  A9 07     LDA #$07
. 1702  8D 47 3F  STA $3F47
. 1705  A9 00     LDA #$00
```

```
. 1707   8D 48 3F STA $3F48
. 170A   A9 FF    LDA #$FF
. 170C   20 D2 FF JSR $FFD2
. 170F   A9 00    LDA #$00
. 1711   8D 40 3F STA $3F40
. 1714   8D 41 3F STA $3F41
. 1717   A9 80    LDA #$80
. 1719   8D 4A 3F STA $3F4A
. 171C   A9 00    LDA #$00
. 171E   8D 49 3F STA $3F49
. 1721   AD 48 3F LDA $3F48
. 1724   18       CLC
. 1725   6D 49 3F ADC $3F49
. 1728   8D 44 3F STA $3F44
. 172B   20 79 17 JSR $1779
. 172E   EE 49 3F INC $3F49
. 1731   AD 49 3F LDA $3F49
. 1734   CD 47 3F CMP $3F47
. 1737   D0 E8    BNE $1721
. 1739   AD 4A 3F LDA $3F4A
. 173C   20 D2 FF JSR $FFD2
. 173F   EE 40 3F INC $3F40
. 1742   AD 40 3F LDA $3F40
. 1745   D0 03    BNE $174A
. 1747   EE 41 3F INC $3F41
. 174A   C9 40    CMP #$40
. 174C   D0 C9    BNE $1717
. 174E   AD 41 3F LDA $3F41
. 1751   C9 01    CMP #$01
. 1753   D0 C2    BNE $1717
. 1755   A9 FF    LDA #$FF
. 1757   20 D2 FF JSR $FFD2
. 175A   A9 0D    LDA #$0D
. 175C   20 D2 FF JSR $FFD2
. 175F   AD 48 3F LDA $3F48
. 1762   18       CLC
. 1763   69 07    ADC #$07
. 1765   8D 48 3F STA $3F48
. 1768   C9 C4    CMP #$C4
. 176A   D0 05    BNE $1771
. 176C   A9 04    LDA #$04
. 176E   8D 47 3F STA $3F47
. 1771   AD 44 3F LDA $3F44
. 1774   C9 C7    CMP #$C7
. 1776   D0 92    BNE $170A
. 1778   60       RTS
. 1779   AD 40 3F LDA $3F40
. 177C   29 F8    AND #$F8
. 177E   8D 42 3F STA $3F42
. 1781   AD 40 3F LDA $3F40
. 1784   29 07    AND #$07
. 1786   8D 43 3F STA $3F43
. 1789   AD 44 3F LDA $3F44
. 178C   4A       LSR
. 178D   4A       LSR
. 178E   4A       LSR
. 178F   8D 45 3F STA $3F45
. 1792   AD 44 3F LDA $3F44
. 1795   29 07    AND #$07
. 1797   8D 46 3F STA $3F46
. 179A   A9 20    LDA #$20
. 179C   85 04    STA $04
. 179E   A9 00    LDA #$00
```

```
. 17A0   85 03      STA $03
. 17A2   AC 45 3F   LDY $3F45
. 17A5   F0 10      BEQ $17B7
. 17A7   A5 03      LDA $03
. 17A9   18         CLC
. 17AA   69 40      ADC #$40
. 17AC   85 03      STA $03
. 17AE   A5 04      LDA $04
. 17B0   69 01      ADC #$01
. 17B2   85 04      STA $04
. 17B4   88         DEY
. 17B5   D0 F0      BNE $17A7
. 17B7   A5 03      LDA $03
. 17B9   18         CLC
. 17BA   6D 42 3F   ADC $3F42
. 17BD   85 03      STA $03
. 17BF   A5 04      LDA $04
. 17C1   6D 41 3F   ADC $3F41
. 17C4   85 04      STA $04
. 17C6   AD 46 3F   LDA $3F46
. 17C9   18         CLC
. 17CA   65 03      ADC $03
. 17CC   85 03      STA $03
. 17CE   A5 04      LDA $04
. 17D0   69 00      ADC #$00
. 17D2   85 04      STA $04
. 17D4   A9 80      LDA #$80
. 17D6   AE 43 3F   LDX $3F43
. 17D9   F0 04      BEQ $17DF
. 17DB   4A         LSR
. 17DC   CA         DEX
. 17DD   D0 FC      BNE $17DB
. 17DF   21 03      AND ($03,X)
. 17E1   F0 11      BEQ $17F4
. 17E3   A9 01      LDA #$01
. 17E5   AC 49 3F   LDY $3F49
. 17E8   F0 04      BEQ $17EE
. 17EA   0A         ASL
. 17EB   88         DEY
. 17EC   D0 FC      BNE $17EA
. 17EE   0D 4A 3F   ORA $3F4A
. 17F1   8D 4A 3F   STA $3F4A
. 17F4   60         RTS
```

Questa routine usa alcuni byte come memoria di lavoro:

.$3F40,$3F41: contatore per X (byte basso/alto).
.$3F42: INT(X/8)*8 (byte basso).
.$3F43: X AND 7.
.$3F44: Y.
.$3F45: INT(Y/8).
.$3F46: Y AND 7.
.$3F47: numero di linee che compongono la striscia corrente (normalmente 7; per l'ultima striscia 4).
.$3F48: 7 per numero di strisce gia' stampate.
.$3F49: numero di linea nella striscia.
.$3F4A: codice del carattere da stampare.

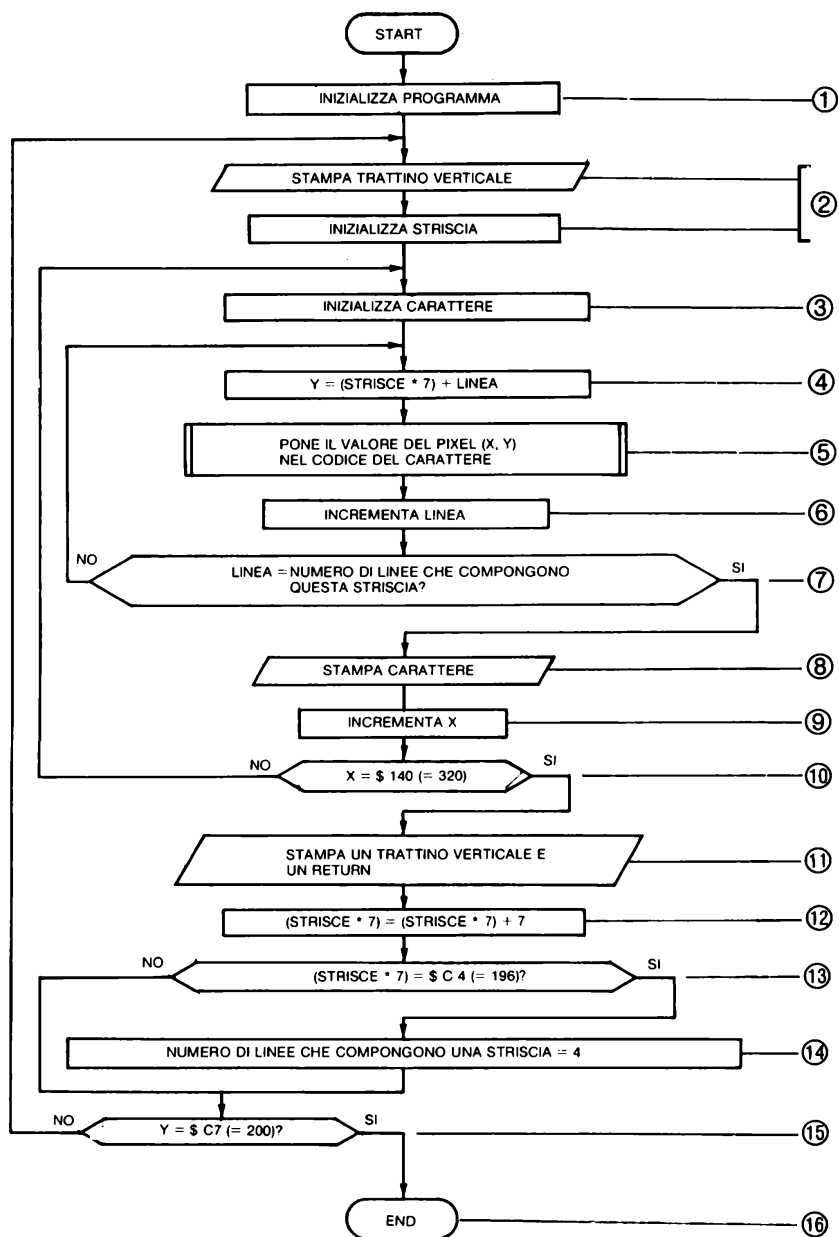



Figura 2.5 Diagramma a blocchi sottoprogramma HARD4

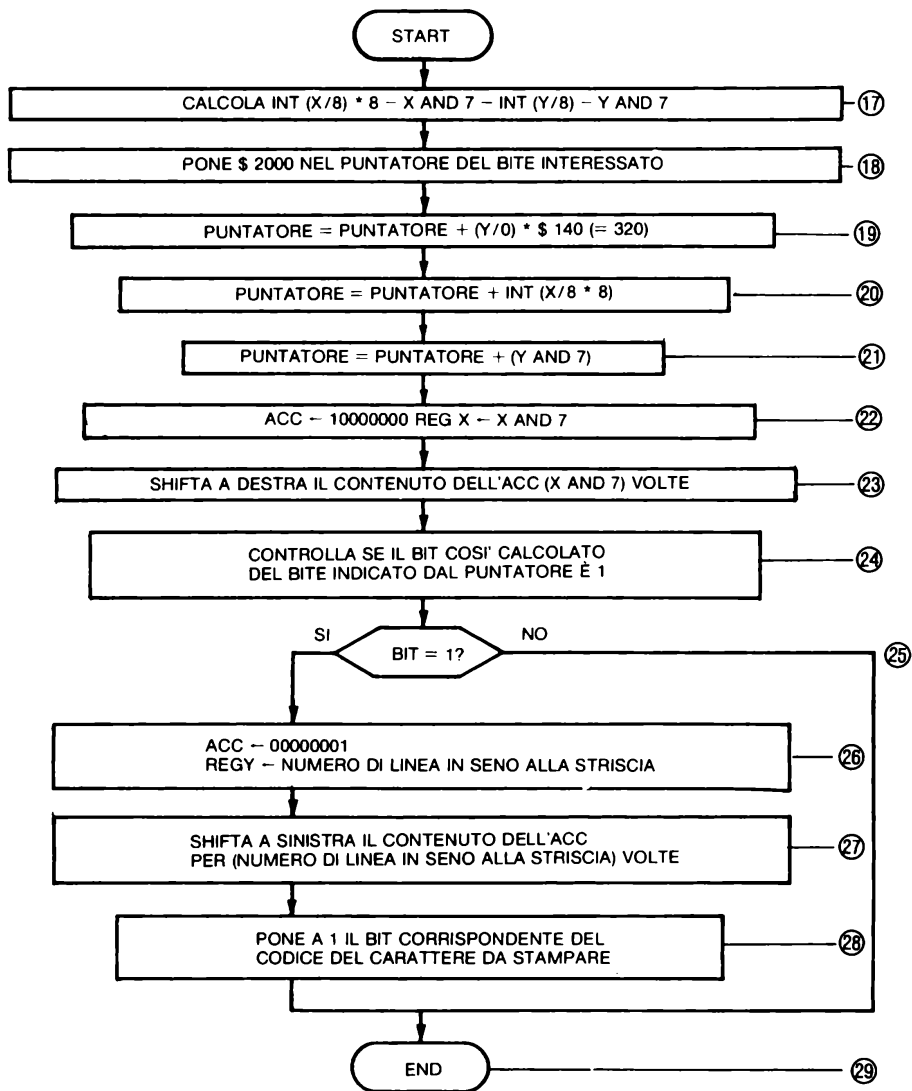



Figura 2.6 Diagramma a blocchi sottoprogramma interno a HARD4

Dal momento che questo sottoprogramma non e'molto semplice, riportiamo nelle Figure 2.5 e 2.6, rispettivamente il diagramma a blocchi di HARD4 e quello della routine interna ad esso.

Per facilitare la comprensione del programma, abbiamo numerato ogni blocco dei diagrammi (con un numero posto fuori a destra) e diamo la corrispondenza tra i blocchi e le linee del listato ASSEMBLER:

Per la Figura 2.5:
BLOCCO 1 : linee 1700-1707.
BLOCCO 2 : linee 170A-1714.
BLOCCO 3 : linee 1717-171E.
BLOCCO 4 : linee 1721-1728.
BLOCCO 5 : linea 172B.
BLOCCO 6 : linea 172E.
BLOCCO 7 : linee 1731-1737.
BLOCCO 8 : linee 1739-173C.
BLOCCO 9 : linee 173F-1747.
BLOCCO 10 : linee 174A-1753.
BLOCCO 11 : linee 1755-175C.
BLOCCO 12 : linee 175F-1765.
BLOCCO 13 : linee 1768-176A.
BLOCCO 14 : linee 176C-176E.
BLOCCO 15 : linee 1771-1776.
BLOCCO 16 : linea 1778.

Per la Figura2.6:
BLOCCO 17 : linee 1779-1797.
BLOCCO 18 : linee 179A-17A0.
BLOCCO 19 : linee 17A2-17B5.
BLOCCO 20 : linee 17B7-17C4.
BLOCCO 21 : linee 17C6-17D2.
BLOCCO 22 : linee 17D4-17D6.
BLOCCO 23 : linee 17D9-17DD.
BLOCCO 24 : linee 17DF.
BLOCCO 25 : linee 17E1.
BLOCCO 26 : linee 17E3-17E5.
BLOCCO 27 : linee 17E8-17EC.
BLOCCO 28 : linee 17EE-17F1.
BLOCCO 29 : linee 17F4.

Per poter usare con soddisfazione VIDEOGRAF, devi prima aver disegnato qualcosa sulla pagina grafica, ed essere tornato in modo testo. Noi abbiamo usato il programma ES7.1, riportato nel primo volume a pag. 90, leggermente modificato. Esso infatti lasciava sporca la prima posizione carattere del video grafico, anche se questo non era visibile dato il colore usato. Il programma modificato si chiama SCRIVEGRAF.

```
0 REM SCRIVEGRAF
5 COLOR0,1:COLOR4,1:YY=1:NC=40:C=8:TRAP500
10 GRAPHIC1,1
20 GETKEYA$
30 IFA$=CHR$(13)THENXX=0:YY=YY+1:GOTO20
40 IFA$=CHR$(20)THENGOSUB300:GOTO20
50 IFA$=CHR$(27)THENGOSUB400:GOTO20
60 GOSUB200
70 XX=XX+C:IFXX>319-CTHENXX=0:YY=YY+1
80 GOTO20
200 COLOR1,1:CHAR,0,0,A$:COLOR1,7,7
210 FORX=0TOC-1
220 FORY=0TO7
230 LOCATEX*8/C,Y
240 DRAWRDOT(2),XX+X,YY*8+Y
250 NEXTY,X
260 RETURN
300 XX=XX-C:YY=YY+(XX<0):XX=XX-320*(XX<0)
310 A$=" ":GOSUB200:RETURN
400 GETKEYA$,B$:NC=VAL(A$)*10+VAL(B$)
410 C=320/NC:RETURN
500 FORX=0TO7:POKE(8192+X),0:NEXTX:GRAPHIC0:SCNCLR
```

E' stato cambiato il nome e aggiunto qualcosa alla linea 500. Il tasto ESC seguito da un numero modifica il numero dei caratteri di una linea video.
Con SCRIVEGRAF abbiamo preparato il disegno che vedi riprodotto da HARD4. Ovviamente puoi usare qualunque programma per produrre un disegno in pagina grafica.

RISULTATO VIDEOGRAF

```
QUADRO VIDEO PER PROVA HARDCOPY
DEL VIDEO GRAFICO
OTTENUTO CON IL PROGRAMMA
          SCRIVEGRAF

ESC 20
ESC 15
ESC 10
ESC 30
ESC 40
ESC 60
ESC 50
```

CAPITOLO 3

I FILE SU DISCO

3.1 INTRODUZIONE

Al COMMODORE 16 possono essere collegate in serie diverse periferiche tra unita' a disco e stampanti.
Le unita' disco sono di norma vendute con dn=8, cioe' gli switch interni sono posizionati su 8. Se il calcolatore e' collegato a piu' unita', la prima puo' mantenere dn=8, la o le altre possono avere dn da 9 a 11. Per modificare il "dn", si puo' agire all'interno dell'unita', rendendo permanente il nuovo numero, oppure procedere all'assegnazione temporanea via software del nuovo numero, al momento dell'accensione.
Noi ci occupiamo dell'unita' 1541 e i programmi esempio si riferiscono al collegamento di una sola unita'.

L'unita' 1541 e' una periferica intelligente, cioe' essa lavora in modo indipendente dopo aver ricevuto i comandi dal calcolatore. Le sue parti componenti sono:
. un microprocessore 6502,
. 16K di memoria ROM, contenenti il DOS (Disk Operating System),
. 2K di memoria RAM per i buffer e le memorie di lavoro,
. un'interfaccia seriale IEEE488,
. due porte di comunicazione,
. le parti elettromeccaniche necessarie.
Le due porte di comunicazione consentono di collegare in serie piu' unita'.

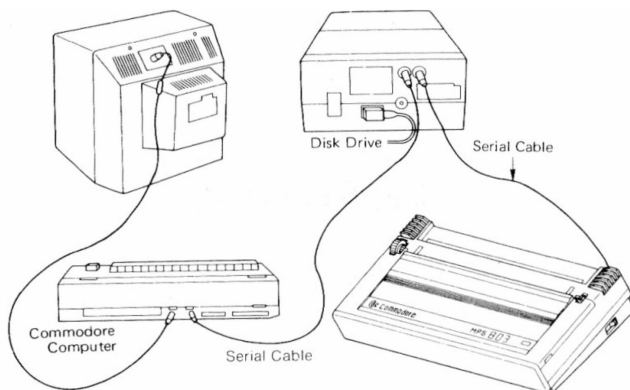



Figura 3.1 Collegamento tra calcolatore, unita' 1541
e stampante MPS-803

Nella parte anteriore dell'unita' sono visibili:
. lo sportellino e la fessura per l'introduzione del dischetto,
. un indicatore luminoso, a sinistra, a luce verde, che indica se l'unita' e' accesa,
. un indicatore luminoso, piu' verso il centro, a luce rossa, che indica, quando acceso, se e' in corso fisicamente un'operazione sul dischetto. Questo indicatore lampeggia quando si verifica un errore.

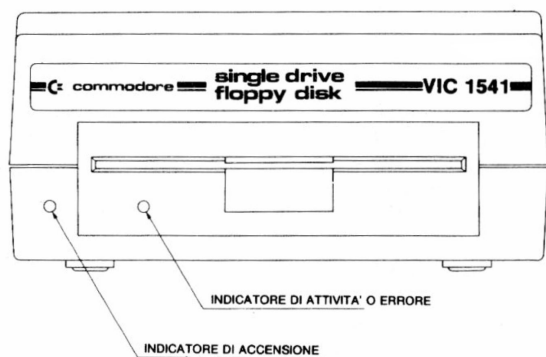



Figura 3.2 Parte anteriore unita' 1541

Il dischetto flessibile, floppy, e' contenuto in una busta protettiva di plastica con alcune aperture, visibili nella Figura 3.3.

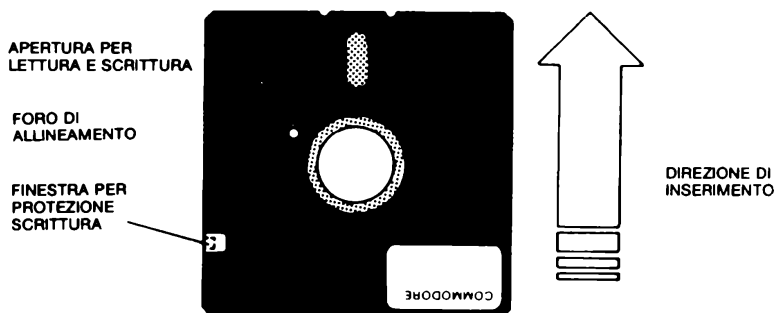



Figura 3.3 Il dischetto nella sua busta

Il dischetto non deve mai essere maneggiato toccando le aperture. La testina di lettura e scrittura agisce attraverso la finestra oblunga centrale. La finestra laterale serve per proteggere il disco da scrittura, basta chiuderla con una delle apposite etichette. Il floppy non deve essere estratto con acceso l'indicatore rosso.

Le registrazioni sul floppy sono eseguite secondo tracce concentriche. Sul tipo di floppy usati per l'unita' 1541 sono disponibili 40 tracce su una sola faccia; di queste ne vengono utilizzate solo 35. Ogni traccia e' divisa in blocchi, chiamati settori, in numero diverso per gruppi di tracce, come sotto riportato.

FORMATO DEL FLOPPY

Numero traccia	Numero settori	Numerazione settori
da 1 a 17	21	da 0 a 20
da 18 a 24	19	da 0 a 18
da 25 a 30	18	da 0 a 17
da 31 a 35	17	da 0 a 16

Facendo i conti risultano disponibili in tutto:
17*21 + 7*19 + 6*18 + 5*17 = 683 settori.

Il settore e' il record fisico del floppy; esso contiene 256 byte. Nel seguito indichiamo con byte 0 il primo e con byte 255 l'ultimo del settore. La traccia 18, di 19 settori, che si trova al centro del dischetto (vedi Figura 3.4), e' utilizzata in un modo particolare; essa contiene l'indice delle registrazioni effettuate sul floppy e la mappa dell'occupazione dei settori. Questo e' necessario dato che il floppy e' un supporto di registrazione ad accesso diretto, non rigorosamente sequenziale come il nastro, e si deve conoscere l'ubicazione delle diverse registrazioni.

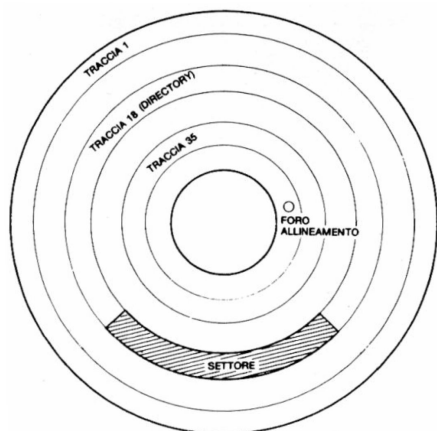



Figura 3.4 Schema non in scala del dischetto

Il dischetto nuovo non puo' essere utilizzato immediatamente; prima e' necessario eseguire su di esso un'operazione di preparazione all'uso, che si chiama "formattazione". Essa consiste nella registrazione sul dischetto degli indirizzi di traccia e settore, nell'assegnazione di un nome e di una identificazione di 2 caratteri, nella creazione della "mappa" del dischetto e della "directory".

Vediamo l'utilizzo della traccia 18 e dei suoi 19 settori, di indirizzo da 0 a 18.

TRACCIA 18 - SETTORE 0

. byte da 0 a 143, mappa di occupazione, chiamata BAM (Block Availability Map).

Questi 144 byte sono utilizzati cosi':
. byte 0 e 1, concatenamento al settore seguente, contengono 18 (12H) e 1 (01H), traccia 18 settore 1.
. byte 2, contiene 65, codice ASCII della lettera A, che indica il formato dell'unita'.
. byte 3, contiene tutti bit 0 e non e' usato.
. byte da 4 a 143, 140 byte che contengono la mappa di occupazione, utilizzando 4 byte per ogni traccia (35*4=140). Ogni quartina di byte e' utilizzata cosi':
. primo byte, numero dei settori ancora disponibili nella traccia;
. secondo byte, mappa dei primi 8 settori della traccia, utilizzando i bit a partire dal meno significativo; quello piu' a destra per il settore 0, quello piu' a sinistra per il settore 7;
. terzo byte, mappa degli 8 settori seguenti, da 8 a 15;
. quarto byte, mappa degli ultimi settori della traccia, a partire dal 16. Restano inutilizzati alcuni bit a seconda delle tracce.
Se il bit e' 1, il settore corrispondente e' libero, se e' 0, esso e' occupato.
. byte da 144 a 255, identificazione del dischetto, blocco di inizio della directory:
. byte da 144 a 161, 18 byte, di cui 16 per il nome, se esso e' piu' corto viene completato con il codice ASCII 160, corrispondente al carattere SHIFT-spazio, gli ultimi 2 byte sempre con codice 160;
. byte 162 e 163, i due caratteri di identificazione (ID) del floppy;
. byte 164, codice 160;
. byte 165 e 166, caratteri 2A, versione del DOS e formato dischetto;
. byte da 167 a 170, codice 160;
. byte da 171 a 255, non utilizzati, con tutti i bit a 0.

TRACCIA 18 - SETTORI DA 1 A 18

Contengono l'indice del disco "directory"; in ogni settore e' registrato l'indice di 8 file. In conseguenza il numero massimo di file registrabili e' 144, 18*8=144. Ogni registrazione nell'indice viene chiamata "entrata" (entry), e occupa 30 byte. La struttura di ogni settore e' la seguente:
. byte 0 e 1, concatenamento al settore seguente. Se il settore e' l'ultimo della catena, il byte 0 contiene tutti bit 0 e il byte 1 contiene il puntatore

all'ultimo byte utilizzato, quindi 255 se il settore e' completo.

. byte da 2 a 31, entrata 1:

. primo byte, tipo del file:
tutti bit 0 per DEL, file cancellato,
128 + 1 per SEQ, file sequenziale,
128 + 2 per PRG, file programma,
128 + 3 per USR, file utente,
128 + 4 per REL, file relativo.

. secondo e terzo byte, indirizzo di traccia e settore del primo blocco del file.

. dal quarto al 19-esimo byte, 16 byte per il nome del file, completato con codice 160 se piu' corto.

. 20-esimo e 21-esimo byte, usati per i file relativi, contengono gli indirizzi di traccia e settore del primo blocco "side sector" (indice interno del file relativo).

. 22-esimo byte, usato solo per i file relativi, contiene la lunghezza del record, <=254.

. dal 23-esimo al 26-esimo byte, 4 byte non usati.

. 27-esimo e 28-esimo byte, traccia e settore del primo blocco del file rimemorizzato, cioe' del file che e' stato memorizzato usando il carattere "@".

. 29-esimo e 30-esimo byte, numero dei blocchi occupati dal file, nell'ordine LO-HI.

. byte 32 e 33, 2 caratteri separatori, contenenti tutti bit 0.

. byte da 34 a 63, entrata 2.

. ...

. ...

. byte 224 e 225, 2 caratteri separatori.

. byte da 226 a 255, entrata 8.

Dei 683 settori disponibili, 19 sono usati per le informazioni sopra descritte; in conseguenza restano disponibili per i file 664 settori (683-19=664).

Dopo l'inserimento del dischetto nell'unita' 1541, al primo accesso avviene l'operazione di "inizializzazione", che consiste nel:

. allineare la testina di lettura con la traccia,

. leggere il settore 0 della traccia 18 in uno dei buffer di 256 byte dell'unita'. Questa operazione e' essenziale, infatti per poter scrivere sul dischetto deve essere consultabile la BAM, altrimenti si rischia di scrivere sopra settori gia' occupati. Inoltre la

BAM deve essere riscritta sul dischetto, prima di toglierlo dall'unita', altrimenti il dischetto non e' piu' utilizzabile in modo corretto.
Devi fare attenzione e non confondere la formattazione con l'inizializzazione; le differenze tra le due operazioni sono:

. la formattazione agisce sul dischetto e lo modifica,

. l'inizializzazione preleva informazioni dal dischetto e le carica nella RAM dell'unita' 1541.

La directory, invece, viene letta a pezzi quando serve, settore per settore, per trovare la localizzazione di un file e le informazioni necessarie alla sua gestione. Essa viene parzialmente riscritta per registrare le modifiche, quando serve.
Abbiamo detto che in un settore si possono registrare 256 byte; in realta' il settore e' formato anche da altri byte, ma questi sono utilizzati dal sistema, per l'indirizzamento e per i controlli. Tu puoi usare solo i 256 byte disponibili per l'utente.

Le caratteristiche tecniche del floppy sono:

.Capacita' totale: 174848 byte (683*256)

.Capacita' per file sequenziali 168656 byte (in ogni settore i primi 2 byte sono usati per il concatenamento, quindi 254*664)

.Capacita' per file relativi 167132 (664*256 - 664*2(concatenamento) - 254*6(side sector)), con al massimo 65535 record logici per file

.Entrate nella directory: 144

.Settori per traccia: da 17 a 21

.Byte per blocco: 256

.Tracce: 35

.Totale blocchi: 683, di cui disponibili 664 per registrare file.

Le operazioni di lettura e scrittura relative al floppy avvengono sempre a livello di blocco fisico, cioe' di un intero settore.

3.2 IL DOS

Il DOS (Disk Operating System) e' una versione del sistema operativo DOS residente nella ROM dell'unita' 1541; esso esegue i comandi che riceve dal calcolatore, funzionando in modo indipendente. Il COMMODORE 16 invia attraverso l'interfaccia seriale sequenze di

byte alla periferica 1541; il DOS riconosce se si tratta di comandi o di dati e gestisce le operazioni disco.

Nel Paragrafo 1.1 abbiamo riepilogato le istruzioni BASIC per la gestione dei file; ora vediamo il significato dei parametri per la periferica 1541.

. lfn, numero logico del file puo' variare da 0 a 127.

. dn, numero dell'apparecchiatura, vale 8 se si usa una sola unita' collegata. Puo' variare da 8 a 11 per piu' unita' collegate.

. sa, secondo la terminologia COMMODORE viene in questo caso chiamato "canale" invece di "indirizzo secondario". Esso gioca un ruolo molto importante, infatti il suo valore permette al DOS di interpretare i dati trasmessi. I valori possibili per "sa" vanno da 0 a 15:

. sa=15, canale comandi;

. sa=0, canale usato per aprire un file in lettura;

. sa=1, canale usato per aprire un file programma in scrittura;

. sa=2,...,14, canale per trasferimento dati.

. nomef, nome del file puo' essere lungo fino a 16 caratteri. Esso puo' essere preceduto dal numero del drive, che e' 0 e puo' essere omesso per unita' singole, deve essere 0 o 1 per unita' doppie.

. tipo, tipo del file, puo' essere: PRG, SEQ, USR, REL (abbreviato alla prima lettera).

. modo, puo' essere W per scrivere e R per leggere, nel caso dei file sequenziali. Si puo' usare anche A (Append) con il significato di prolungare (aggiungere in coda dati) un file sequenziale.

Attraverso il canale 15, il calcolatore invia all'unita' 1541, con le istruzioni OPEN e/o PRINT#, stringhe di comandi, che esaminiamo nel seguito. Il DOS provvede a interpretare i comandi e li manda in esecuzione; esso assegna al canale lo spazio di memoria RAM necessario per lavorare e i buffer necessari. La RAM dell'unita' 1541 comprende 8 buffer di 256 byte ciascuno; di essi 4 sono sempre impegnati per la BAM, le variabili di lavoro e il controllo delle operazioni. Restano a disposizione solo 4 buffer per i file; in conseguenza si possono gestire contemporaneamente 3 o 4 file, a seconda del loro tipo.

Prendiamo ora in esame un primo gruppo di stringhe di

comandi per la gestione del dischetto; esse si possono inviare al DOS in BASIC, sia in modo immediato che da programma. Per inviare il comando si possono seguire due strade:

.1) OPENlfn,8,15
    PRINT#lfn,stringa-comandi

.2) OPENlfn,8,15,stringa-comandi

e in ambedue i casi devi terminare con CLOSElfn. Se non si esegue la CLOSE, ad una successiva apertura si ha errore. Puoi decidere di aprire all'inizio del programma il canale 15, di inviare i comandi con PRINT#lfn e di chiudere il canale solo alla fine del programma.

I comandi di utilita' generale disponibili, che si possono scrivere per esteso o abbreviandoli alla prima lettera, sono:

.NEW, per formattare il dischetto,
.INITIALIZE, per allineare la testina e caricare la BAM,
.VALIDATE, per sistemare la BAM in base alle entrate valide registrate nella directory,
.COPY, per copiare file,
.RENAME, per cambiare nome a un file,
.SCRATCH, per cancellare un file.

Il BASIC 3.5 del COMMODORE 16 mette a disposizione, come gia' visto nel primo volume, alcuni comandi che producono lo stesso effetto di alcuni di questi, e non hanno bisogno di essere preceduti dalla OPEN del canale 15.

NEW
puo' essere usato per:
.preparare un disco nuovo (o gia' usato) per l'uso. Vengono registrati gli indirizzi di traccia e settore, il nome e l'identificazione, viene preparata la BAM.
.cancellare un disco gia' usato, mantenendo invariata l'dentificazione, ma senza riscrivere gli indirizzi, e aggiornando la BAM.
La stringa si scrive: "Ndr:nome,XX"
.dr, puo' essere omesso se 0.

.nome, e' il nome da assegnare al dischetto, massimo 16 caratteri.
.XX, e' la "identificazione" del dischetto; deve essere di 2 caratteri (scelti a piacere). Omettendo XX con un floppy gia' usato, si ottiene di cancellare il precedente contenuto, senza riscrivere gli indirizzi. Non puoi omettere XX con un floppy nuovo.

Il comando BASIC equivalente e' HEADER.

INITIALIZE
deve essere usato per allineare la testina di lettura e scrittura all'inizio della traccia e per caricare la BAM in un buffer dell'unita' 1541.
La stringa si scrive: "Idr"
.dr, puo' essere omesso se 0.
Quando inserisci il floppy nell'unita' questa operazione deve avvenire automaticamente, comunque e' bene eseguirla quando si usa molto il dischetto. Prima di eseguirla devi chiudere i file eventualmente aperti.

Non esiste un comando BASIC equivalente.

VALIDATE
serve per rimettere in ordine un dischetto nel quale la situazione della BAM non corrisponde alle entrate della directory, ci sono file non correttamente chiusi (con un asterisco di fianco al tipo nella lista della directory). Devi eseguire questa operazione quando la somma dei blocchi occupati e dei blocchi liberi non da' 664.
La stringa si scrive: "Vdr"
.dr, puo' essere omesso se 0.
Il comando rigenera la BAM in base alle entrate della directory e cancella le entrate della directory non valide.

Il comando BASIC equivalente e' COLLECT.

COPY
consente di ottenere copie di un file di qualunque tipo. Se il floppy e' unico (come per l'unita' 1541) il nome vecchio e il nome nuovo devono essere diversi. Si possono anche fondere piu' file sequenziali di

dati in un unico file. Lavora su unita' singola o su unita' a due drive, non su due unita' diverse.
La stringa si scrive:
"Cdr:dnomef=dr:snomef" per copia di un file
"Cdr:dnomef=dr:snomef1,snomef2,..." per fondere file sequenziali di dati.
.dnomef, e' il file destinazione.
.snomef, e' il file sorgente.
Il file sorgente deve essere stato correttamente chiuso.
Il comando BASIC equivalente e' COPY.
Inoltre per unita' a due floppy in BASIC e' disponibile anche il comando BACKUP (DUPLICATE del DOS).
Per il momento non e' disponibile un'unita' a 2 drive da collegare direttamente al COMMODORE 16.

==

RENAME
serve per cambiare nome a un file nella directory senza spostare il file.
La stringa si scrive: "Rdr:nnomef=vnomef"
.nnomef, e' il nome nuovo.
.vnomef, e' il nome vecchio.
Il file su cui operi deve essere stato correttamente chiuso.

Il comando BASIC equivalente e' RENAME.

==

SCRATCH
cancella uno o piu' file ponendo DEL (tutti bit 0) come tipo nell'entrata della directory e aggiorna la BAM.
La stringa si scrive:
"Sdr:nomef" per un solo file
"Sdr:nomef1,nomef2,.." per piu' file.
Ti consigliamo di cancellare i file che non ti servono piu' prima di riscriverli e di non usare il carattere "@" nella OPEN o nelle SAVE, DSAVE. Infatti quest'ultimo modo di procedere puo', a volte, generare degli inconvenienti sul dischetto.

Il comando BASIC equivalente e' SCRATCH.

Per tutti i comandi visti il "dn" viene citato nella OPEN e seleziona l'unita', se ci sono piu' collegamenti.

Oltre a quelli gia' esaminati sono disponibili altri tre gruppi di comandi che servono per:

.gestire direttamente le operazioni di lettura e scrittura sul dischetto accedendo a un blocco fisico, individuato dal suo indirizzo di traccia e settore.

.gestire i file relativi.

.aggiungere al DOS routine utente, che possono essere lette dal floppy e scritte sul floppy e mandarle in esecuzione.

GESTIONE DIRETTA

Esaminiamo ora i comandi per la gestione diretta delle operazioni sul floppy. Per poter eseguire questi comandi deve essere aperto il canale 15 e deve essere aperto uno dei canali, da 2 a 14, per il trasferimento dei dati.
Nel seguito presupponiamo che siano state eseguite le due OPEN, prima della PRINT# che invia il comando, e che alla fine delle operazioni vengano eseguite le due corrispondenti CLOSE.
Le OPEN devono essere eseguite nel seguente ordine:

. OPENlfn,8,15 per aprire il canale comandi con un determinato "lfn". Noi, per abitudine usiamo il numero 15 per questo "lfn", cosi' non dobbiamo ricordare un altro numero: OPEN15,8,15.

. OPENlfn,8,sa,"#" per aprire il canale per i dati. Lo "lfn" usato qui deve essere diverso da quello della OPEN precedente; esso deve variare tra 0 e 127; "sa" puo' variare da 2 a 14. Noi di solito usiamo per "lfn" e "sa" valori uguali, compresi tra 2 e 14, ma escludiamo il 4, dato che, per abitudine, lo usiamo per la stampante. Il carattere "#" deve essere presente e significa apertura per accesso diretto. I buffer dell'unita' 1541 sono numerati da 0 a 7; se vuoi puoi scrivere dopo il carattere "#" un numero per scegliere un buffer particolare, con il rischio di trovarlo gia' occupato. Se scrivi "#4", chiedi di usare il quinto buffer per i dati. Qualora desideri sapere quale dei buffer il sistema ha usato, puoi subito dopo la OPEN..."#", eseguire GET#lfn,A$; in A$ trovi il numero del buffer assegnato. Questa ultima OPEN non produce una registrazione nella directory; questo fatto puo' essere pericoloso, infatti, se si eseguono i comandi VALIDATE o COLLECT, le registrazioni dirette spariscono, nel senso che vengono liberati i settori della BAM non registrati nella directory.

Al termine delle operazioni la CLOSE del canale dei dati deve essere eseguita prima della CLOSE del canale comandi. Il canale comandi puo' essere aperto all'inizio del programma e chiuso alla fine.

I comandi disponibili in questo gruppo sono:

. BLOCK-ALLOCATE, per allocare un blocco, abbreviato in B-A;
. BLOCK-FREE, per liberare un blocco, abbreviato in B-F;
. BLOCK-READ, per leggere un blocco nel buffer dei dati, abbreviato in B-R;
. BLOCK-WRITE, per scrivere il contenuto del buffer dei dati sul disco;
. BUFFER-POINTER, per posizionare il puntatore in una determinata posizione nel buffer dei dati;
. USER1, simile a B-R, abbreviato in U1;
. USER2, simile a B-W, abbreviato in U2.

Nelle spiegazioni che seguono usiamo i parametri gia' noti, e, in piu', "t" per traccia e "s" per settore, "p" per posizione del puntatore nel buffer.
Precisiamo come si svolgono le operazioni dirette sul floppy:
.SCRITTURA: le istruzioni PRINT#lfn,lista-dati scrivono nel buffer dei dati nella RAM dell'unita' 1541, facendo avanzare il puntatore; per trasferire il blocco sul dischetto si usa l'apposito comando trasmesso sul canale 15, con PRINT#.
.LETTURA: il blocco viene trasferito nel buffer dei dati nella RAM dell'unita' 1541 dall'apposito comando, trasmesso sul canale 15, con PRINT#; le istruzioni INPUT#lfn,lista-dati e GET#lfn,lista-dati leggono i dati dal buffer, facendo avanzare il puntatore.
La posizione del puntatore nel buffer puo' essere controllata, usando il relativo comando, trasmesso sul canale 15.

==

BLOCK-ALLOCATE, si scrive:
"B-A:"dr;t;s
registra nella BAM l'occupazione del blocco di indirizzo t,s. E' importante allocare un settore libero, altrimenti si cancella qualcosa. Per essere sicuri che un settore e' libero si chiede di allocarlo e subito dopo si analizza la situazione di errore tramite il

canale 15. L'errore 65: NO BLOCK, segnala che il settore e' gia' occupato. In questo caso le due variabili ET e ES, fornite dall'analisi dell'errore, danno l'indirizzo t,s del primo settore disponibile, che puo' essere allocato. Se, invece, ET contiene 0, questo significa che il dischetto e' pieno. EN contiene 0 se l'allocazione del blocco non ha creato problemi. E' importante tener presente che il sistema non evita di allocare settori eventualmente liberi nella traccia 18 della directory; per questa ragione devi controllare con il tuo programma di non andare ad occuparli. Infatti un'eventuale estensione della directory per nuove entrate potrebbe poi danneggiare le tue registrazioni (vedi routine ALLOCA nel Paragrafo 3.8). Vedi NOTA al comando seguente.

BLOCK-FREE, si scrive:

"B-F:"dr;t;s

libera il blocco di indirizzo t,s e aggiorna la BAM.

NOTA: B-A e B-F lavorano anche se prima non e' stata eseguita una OPEN..."#", ma questo e' pericoloso, infatti la BAM viene riscritta sul floppy quando si chiude un canale per la trasmissione dei dati; se esso non e' stato aperto si puo' danneggiare il dischetto.

BLOCK-READ, si scrive:

"B-R:"sa;dr;t;s

trasferisce il settore di indirizzo t,s nel buffer dei dati assegnato al momento della OPEN..."#" con lo stesso valore di "sa". Nella posizione 0 del buffer (primo carattere) viene trovato il valore del puntatore (fine record) al momento della registrazione del blocco. Tale indicazione e' importante perche' in fase di lettura dal buffer con le istruzioni INPUT# o GET# la parola di stato ST contiene 64 quando si raggiunge la posizione di fine record.

BLOCK-WRITE, si scrive:

"B-W:"sa;dr;t;s

scrive il contenuto del buffer dei dati, assegnato al momento della OPEN..."#" con lo stesso valore di "sa", nel blocco individuato dall'indirizzo t,s. Registra nella posizione 0 il valore raggiunto dal puntatore e predispone il puntatore sulla posizione 1. Se il bloc-

co viene letto con B-R, il primo carattere disponibile e' quello di posizione 1, mentre, se la lettura viene effettuata con U1, il primo carattere disponibile e' quello di posizione 0.

===

BUFFER-POINTER, si scrive:

"B-P:"sa;p

consente di spostare il puntatore all'interno del buffer dei dati. La posizione del puntatore e' molto importante, infatti le istruzioni di I/O agiscono a partire dalla sua posizione. Nelle due coppie di istruzioni che scrivono o leggono un settore abbiamo indicato dove si trova il puntatore dopo la loro esecuzione. Questo comando consente di controllare completamente la posizione del puntatore.
Il sistema gestisce i file sequenziali usando i primi due caratteri di ogni blocco, di posizione 0 e 1, per concatenare tra loro i settori; quando usi i file random puoi mantenere questa organizzazione preoccupandoti tu di riempire i primi due caratteri e posizionando i dati a partire dalla posizione 2. In sostanza in un settore possono cosi' essere utilizzati solo 254 byte. Lo spostamento del puntatore consente di raggiungere qualunque campo nel record di un file random.

===

USER1, si scrive:

"U1:"sa;dr;t;s

agisce come B-R, con la differenza che legge tutto il settore, senza tener conto della posizione del puntatore al momento della scrittura. Dopo l'esecuzione di U1 il puntatore si trova nella posizione 0 dell buffer.

===

USER2, si scrive:

"U2:"sa;dr;t;s

agisce come B-W, ma non registra in posizione 0 la posizione finale del puntatore.

Nella descrizione delle stringhe comando noi abbiamo usato il formato: comando abbreviato, seguito da due punti, tra virgolette, e dopo i parametri separati da ";". E' possibile scrivere anche tutto tra virgolette, nel caso che i parametri siano costanti, usan-

do il separatore virgola; per esempio, cosi':
"U2:3,0,15,8"
I comandi possono essere scritti anche in modo non abbreviato.

GESTIONE FILE RELATIVI

I file relativi sono formati da record logici di lunghezza fissa. Il DOS li gestisce ricevendo una stringa comando che indica il numero d'ordine del record logico nel file e la posizione del puntatore nel record logico.
Anche in questo caso sono necessarie 2 istruzioni OPEN:
. OPENlfn,8,15, per aprire il canale comandi.
. OPENlfn,8,sa,nomef+",L,"+CHR$(LU), per aprire il canale dei dati con "lfn" diverso da quello del canale comandi, "sa" compreso tra 2 e 14, L parametro richiesto per segnalare che l'apertura riguarda un file relativo, seguito dalla lunghezza del record logico passata come stringa. La lunghezza del record logico e' al massimo 254. Questa OPEN produce la registrazione di un'entrata per il file nella directory.
Al termine delle operazioni la CLOSE del canale dei dati deve essere eseguita prima di quella relativa al canale dei comandi.
Il comando disponibile, da trasmettere con una PRINT# sul canale 15, prima di ogni operazione di lettura o scrittura relativa al buffer, serve per selezionare il record logico e per posizionare il puntatore al suo interno.

Esso si scrive:
"P"+CHR$(sa)+CHR$(LO)+CHR$(HI)+CHR$(BI)
.P, significa puntatore.
.CHR$(sa), fornisce il numero del canale usato nella OPEN del canale dei dati, come stringa.
.CHR$(LO)+CHR$(HI), forniscono il numero del record logico che si vuole trattare nella forma byte basso-byte alto. Si calcolano cosi':
HI=INT(numerorecord/256)
LO=numerorecord-HI*256.
.CHR$(BI), fornisce il puntatore nel record. BI=1 per il primo campo, BI=N per puntare al campo che inizia dopo i primi N-1 caratteri.

Dopo aver selezionato un record e una posizione, devi scrivere con una sola PRINT#,lista-dati tutti i campi che desideri nel record. Se esegui altre PRINT#,lista-dati, senza riposizionarti, vai a scrivere sui record successivi. Un record logico puo' essere scritto a pezzi, ma ogni volta devi riposizionarti al record e al campo desiderato.
Per leggere devi posizionarti al record e al campo desiderato e poi eseguire la INPUT#, lista-dati relativa a tutti i campi del record che ti interessano. Se non ripeti l'operazione di posizionamento con successive INPUT# leggi il file in modo sequenziale, un record dopo l'altro.
L'istruzione GET#, legge carattere per carattere a partire dalla posizione selezionata.

PROGRAMMAZIONE IN LINGUAGGIO MACCHINA

I comandi disponibili in questo gruppo sono:

. BLOCK-EXECUTE, per eseguire un programma in linguaggio macchina, memorizzato in un settore del floppy;
. Memory-Write, per registrare al massimo 34 byte nella RAM dell'unita' 1541;
. Memory-Read, per leggere il contenuto di byte della memoria dell'unita' 1541;
. Memory-Execute, per eseguire codice macchina a partire da un byte della memoria dell'unita' 1541;
. USERi, per saltare a particolari locazioni della memoria dell'unita' 1541 ed eseguire routine in codice macchina.

Questi comandi, salvo BLOCK-EXECUTE, richiedono solo la preventiva apertura del canale 15. Il comando BLOCK-EXECUTE richiede anche l'apertura di un altro canale con OPEN#lfn,8,sa,"#", per rendere disponibile un buffer per i dati. Nelle spiegazioni che seguono, oltre ai parametri gia' noti, usiamo anche:
.i, come numero di riferimento nella tabella dei salti USER.
.adl, byte basso dell'indirizzo del blocco di memoria.
.adh, byte alto dell'indirizzo del blocco di memoria.
.nc, numero caratteri da trasferire, da 1 a 34.
.dati, istruzioni in codice macchina; devi usare la funzione CHR$ con argomento uguale al numero decimale che rappresenta il contenuto di ogni byte.

==
BLOCK-EXECUTE, si scrive:
"B-E:"sa,dr,t,s
consente di caricare in un buffer dell'unita' 1541 il contenuto del blocco di indirizzo t,s, e di mandarlo in esecuzione, come programma in linguaggio macchina, a partire dalla posizione 0. Perche' questo comando possa essere eseguito con successo devono essere verificate le seguenti condizioni:

. la routine in linguaggio macchina deve trovarsi nel settore specificato e la prima istruzione deve stare nel primo byte (posizione 0);
. la routine deve occupare al massimo 256 byte;
. il settore del floppy deve essere stato scritto con il comando U2, per non danneggiare la posizione 0;
. la routine deve terminare logicamente con l'istruzione RTS.

La routine in linguaggio macchina puo' lavorare modificando o leggendo i dati degli altri buffer della RAM dell'unita' 1541.
Il comando puo' anche essere scritto senza abbreviazione, e i parametri, se costanti, possono stare tra le virgolette separati da virgola.

==
Memory-Write, si scrive:
"M-W:"adl adh;nc;dati
consente di registrare, al massimo 34 byte, nella RAM dell'unita' 1541. I parametri adl e adh, che forniscono l'indirizzo di memorizzazione, si devono calcolare; per scrivere all'indirizzo 1794:
HI=INT(1794/256)=7
LO=1794-7*256=1794-1792=2
CHR$(2)+CHR$(7) forniscono la stringa da usare nel comando per "adl adh".
Il parametro "nc", deve essere passato come numero, o come variabile numerica.
I parametri "dati" sono ottenuti usando la funzione CHR$ avente come argomento i valori decimali dei byte da scrivere in memoria. Se con questo comando memorizzi una routine in linguaggio macchina, per eseguirla devi usare il comando Memory-Execute. Nota la differenza con BLOCK-EXECUTE.
Questo comando deve essere scritto in modo abbreviato.

==

Memory-Read, si scrive:
    "M-R:"adl adh
consente di leggere il contenuto del byte di indirizzo "adl adh" dalla memoria dell'unita' 1541.
L'indirizzo deve essere passato con la funzione CHR$ dei byte LO e HI.
Il contenuto del byte indirizzato viene reso disponibile sul canale 15, per leggerlo devi eseguire GET# sul canale 15. Se esegui piu' GET# successive l'indirizzo del byte viene incrementato automaticamente.
Per esempio:

```
OPEN15,8,15
PRINT#15,"M-R:"CHR$(X)CHR$(Y)
FORk=0TO7:GET#15,a$(k):NEXTk
CLOSE15
```

legge nella matrice a$(k) 8 byte a partire dall'indirizzo di memoria 256*Y+X dell'unita' 1541.
Il comando puo'essere scritto solo abbreviato.

==

Memory-Execute, si scrive:
    "M-E:"adl adh
consente di eseguire una routine in codice macchina memorizzata a partire dal byte di indirizzo "adl adh".
Per il parametro indirizzo vale quanto detto per M-R.
Il comando puo' essere scritto solo in modo abbreviato.
Invece di prelevare ed eseguire una routine in codice macchina memorizzata su un settore del floppy, usando il comando BLOCK-EXECUTE, puoi usare prima il comando M-W per memorizzare la routine in RAM e poi M-E per eseguirla.

==

USERi, si scrive:
    "Ui" con i=0,3,4,...,8,9 oppure:
    "Ua" con a=J,C,D,...,H,I
consente di saltare a indirizzi fissi nella memoria dell'unita' 1541. Tali indirizzi sono:

```
U3  o  UC    salto a 1280  (0500H)
U4  o  UD      "   " 1283  (0503H)
U5  o  UE      "   " 1286  (0506H)
U6  o  UF      "   " 1289  (0509H)
U7  o  UG      "   " 1292  (050CH)
U8  o  UH      "   " 1295  (050FH)
U9  o  UI      "   " 65530 (FFFAH)
U0  o  UJ      "   alla routine di accensione.
```

Puoi memorizzare a partire da questi indirizzi, se sono RAM, routine in linguaggio macchina e andarle a eseguire usando i comandi Ui relativi.

3.3 GESTIONE DEGLI ERRORI DISCO

Quando si verifica un errore relativo ad un'operazione dell'unita' 1541 l'indicatore luminoso a luce rossa comincia a pulsare, ma non si ha una segnalazione automatica di errore. Per rilevare gli errori devi inserire nei tuoi programmi una routine di errore, leggendo, attraverso il canale 15, i 4 dati seguenti:

. EN, numero dell'errore, dato numerico;
. EM$, messaggio di errore, dato di tipo stringa;
. ET, numero della traccia interessata, dato numerico;
. ES, numero del settore interessato, dato numerico.

Le 4 variabili citate non sono riservate, puoi usare quelle che vuoi, le nostre sono abbastanza mnemoniche. Puoi anche usare variabili di tipo stringa per tutti i dati.
Supponendo che all'inizio del programma sia stata eseguita l'istruzione: OPEN15,8,15, la routine di errore puo' avere questa forma:

```
5000 INPUT#15,EN,EM$,ET,ES
5002 IFEN=0 THEN RETURN
5004 PRINT"ERRORE: ";EN,EM$,ET,ES
5006 STOP
```

essa lascia aperto il canale 15. Ti raccomandiamo di eseguire il controllo degli errori dopo ogni operazione disco.
Un altro modo per controllare se ha avuto luogo un errore e' quello di leggere le due variabili riservate del BASIC: DS e DS$, dopo l'esecuzione di ogni operazione disco. In DS trovi il numero del messaggio di errore, in DS$ trovi il numero e il messaggio di errore, il numero della traccia e il numero del settore. Con DS$ ottieni le stesse informazioni fornite dalla routine di errore, solo che esse si trovano in una sola stringa. Per analizzare o stampare il contenuto di queste due variabili riservate non e' necessario il canale 15; esse sono automaticamente disponibili per il BASIC.
L'esecuzione della routine di errore o l'analisi del-

le variabili DS e DS$ hanno l'effetto di spegnere l'indicatore di errore se esso e' acceso.
Non devi confondere le segnalazioni di errore, relative ad operazioni disco, fornite dal sistema operativo del COMMODORE 16, con quelle del DOS. Se, per esempio, esegui un'istruzione PRINT# su un file non aperto, la segnalazione di errore dipende dal fatto che il COMMODORE 16 non ha trovato aperto il file nella tabella di gestione, il DOS non c'entra.

Il codice di errore 00 significa che l'operazione e' andata bene; il codice 01 che e' stato cancellato un file.

Inoltre, dopo ogni operazione disco puo' essere analizzata la variabile riservata ST; essa puo' avere i seguenti valori:

- 0 per tutto bene
- 1 per scrittura con tempizzazione errata
- 2 per lettura con tempizzazione errata
- 64 per segnalazione di EOF (End Of File)
- 128 per apparecchiatura non presente

e quindi non fornisce informazioni complete sull'andamento dell'operazione.

MESSAGGI ERRORE DOS

==

20 READ ERROR
il controllore del disco non riesce a trovare la testata del settore richiesto, o l'indirizzo non e' valido o il floppy e' rovinato.

==

21 READ ERROR
il controllore del disco non riesce a trovare il carattere di sincronizzazione sulla traccia richiesta. Puo' essersi verificato un errore di allineamento, un errore hardware o il floppy non essere formattato.

==

22 READ ERROR
per un comando DOS di tipo BLOCK non viene trovato il settore; puo' essere invalido l'indirizzo o danneggiato il floppy.

----

23 READ ERROR
si verifica un errore di CHECKSUM dopo la lettura di un blocco, cioe' i controlli sui caratteri letti non tornano; puo' dipendere da una messa a terra difettosa.

24 READ ERROR
si verifica un errore hardware, puo' dipendere da messa a terra difettosa.

25 WRITE ERROR
i dati registrati non corrispondono al contenuto del buffer.

26 WRITE PROTECT ON
tentativo di scrivere su un floppy con protezione, cioe' con finestrella laterale chiusa.

27 READ ERROR
errore nella lettura di una testata, puo' dipendere da messa a terra difettosa.

28 WRITE ERROR
non viene trovato, dopo la scrittura di un blocco, il carattere di sincronizzazione del blocco seguente. Il floppy puo' essere rovinato o non formattato o verificarsi un errore hardware.

29 DISK ID MISMATCH
non riesce a identificare un floppy, che puo' essere non formattato o rovinato.

30 SYNTAX ERROR
il DOS non riesce a interpretare un comando ricevuto, possono essere errati o mancare alcuni parametri.

31 SYNTAX ERROR
il DOS non riconosce un comando, per esempio esso inizia con uno spazio.

32 SYNTAX ERROR
comando troppo lungo, piu' di 58 caratteri.

33 SYNTAX ERROR
e' stato usato un nome non valido.

34 SYNTAX ERROR
il DOS non riconosce il nome di un file.

39 SYNTAX ERROR
non viene riconosciuto un comando inviato sul canale 15.

50 RECORD NOT PRESENT
tentativo di leggere un file dopo aver raggiunto EOF. Per i file relativi si verifica questo errore nella fase di preestensione e deve essere ignorato.

51 OVERFLOW IN RECORD
si tenta di scrivere in un blocco piu' caratteri di quelli consentiti.

52 FILE TOO LARGE
si scrivono piu' record di quelli previsti su un file relativo.

60 WRITE FILE OPEN
si apre in lettura un file gia' aperto per scrivere e non chiuso.

61 FILE NOT OPEN
si tenta un'operazione su un file non aperto.

62 FILE NOT FOUND
il file richiesto non viene trovato.

63 FILE EXISTS
si cerca di creare un file che esiste gia' e non si e' usato il carattere "@".

64 FILE TYPE MISMATCH
il tipo di operazione fa riferimento a un file di tipo diverso da quello esistente.

65 NO BLOCK
indica che il blocco richiesto con B-A e' gia' occupato, ma fornisce in ET e ES l'indirizzo del primo blocco libero. ET e ES sono a zero se il floppy e' pieno.

66 ILLEGAL TRACK AND SECTOR
si tenta di accedere a settori che non esistono.

67 ILLEGAL SYSTEM T OR S
traccia o settore non consentiti.

70 NO CHANNEL
il canale richiesto e' occupato o non ci sono canali liberi.

71 DIRECTORY ERROR
i controlli non consentono di creare una BAM valida. Probabilmente il dischetto non e' ricuperabile.

72 DISK FULL
indica o che il dischetto e' pieno o che sono esaurite le 144 entrate nella directory.

73 DOS MISMATCH
le versioni del DOS non sono compatibili in scrittura. Si tenta di scrivere su un floppy formattato con un'altra versione.

===

74 DRIVE NOT READY
non c'e' il floppy oppure esso non e' stato formattato o il drive e' guasto.

3.4 FILE DI PROGRAMMA

I programmi sono memorizzati sul dischetto come file di tipo PRG. Nella relativa entrata della directory si trovano tutte le informazioni necessarie per reperire il file, e cioe' l'indirizzo di traccia e settore del primo blocco, il numero dei blocchi occupati, e, se necessario, l'indirizzo del primo blocco per la rimemorizzazione con il carattere "@". Per analizzare il contenuto della directory puoi usare il programma DCOMEFS, riportato nel Paragrafo 3.8. I blocchi nei quali viene registrato un programma sono concatenati tra loro tramite i primi 2 caratteri, che recano il numero della traccia e del settore successivo. L'ultimo blocco reca 0 per numero di traccia e il valore del puntatore all'ultimo byte occupato al posto del numero del settore. Per vedere come il programma viene conservato sul disco puoi usare il programma TRAC/SET, riportato nel Paragrafo 3.8; solo che la lettura del contenuto dei blocchi non e' semplice, dato che si vedono i numeri in esadecimale. Se te ne occupi troverai che il programma termina con 2 byte contenenti tutti bit 0, dopo il byte a 0 che chiude l'ultima istruzione.

Per memorizzare un programma sul floppy sono disponibili le istruzioni SAVE e DSAVE, gia' esaminate nel precedente volume. Queste istruzioni possono essere scritte anche facendo precedere al nome del programma i caratteri "@:" oppure "@dr:", con l'effetto di cancellare la precedente versione del programma. Non ti consigliamo di procedere cosi'; infatti a volte operazioni di questo tipo danneggiano la BAM. Se devi rimemorizzare un programma e' preferibile prima cancellarlo con il comando SCRATCH, usato in una delle due versioni disponibili, e poi memorizzarlo con SAVE, ma senza usare il carattere di cancellazione "@".

Dopo la memorizzazione di un programma e' bene verificare con VERIFY la bonta' della registrazione. Puoi anche richiamare con il comando DIRECTORY la directory

sul video; questo comando e' molto comodo dal momento che non cancella il contenuto della memoria e il programma presente resta invariato. Se invece carichi e listi la directory con:
OPEN15,8,15:LOAD"$",8:LIST il programma viene cancellato.

Per caricare in memoria un programma puoi usare i comandi DLOAD e LOAD. Se li usi in immediato il programma viene solo caricato, mentre se li usi da programma, ottieni anche la partenza, come se venisse eseguito: GOTO prima-linea. In questo caso ottieni il concatenamento dei programmi, ma il sistema non mette a posto il puntatore all'inizio della zona variabili, che varia in dipendenza dalla lunghezza dei programmi. Per operare correttamente il programma concatenato deve avere una prima istruzione che sistema tali puntatori, cosi':

0 POKE 45,PEEK(157):POKE 46,PEEK(158):CLR.
Con opportuni accorgimenti e' possibile eseguire programmi concatenati che abbiano le variabili in comune. Basta che il primo programma sposti il puntatore all'inizio delle variabili in posizione tale che vada bene per il programma piu' lungo e che tutti i programmi usino le stesse variabili con lo stesso significato.

Nel precedente volume abbiamo preso in esame il significato del flag che puo' essere aggiunto in fondo alle istruzioni per memorizzare e caricare i programmi e influisce sulla loro rilocabilita' in memoria.

3.5 FILE SEQUENZIALI DI DATI

I file sequenziali di dati su disco hanno le stesse caratteristiche di quelli su nastro; l'unica differenza rilevante consiste nel fatto che e' possibile aggiornare un file senza doverlo trascrivere tutto in memoria (vedi Paragrafo 11.6 del primo volume), ma creando un nuovo file sullo stesso dischetto, con nome diverso, nel quale trascrivere e aggiornare record dopo record il vecchio file. Al termine dell'aggiornamento il vecchio file puo' essere cancellato, e al nuovo file puo' essere assegnato il vecchio nome con il comando RENAME.
Riepiloghiamo le caratteristiche di questo tipo di file:

. record fisico di dimensioni pari a un settore, con i primi due caratteri usati per il concatenamento dei settori e gli altri 254 per i dati. La gestione del record fisico risulta trasparente per l'utente che lavora a livello di record logico. Alla fine del file viene riconosciuta la condizione EOF; essa puo' essere controllata tramite la parola di stato ST, le variabili riservate DS o DS$, la routine di errore leggendo EN, EM$, ET e ES sul canale 15.

. record logico di lunghezza fissa o variabile, ma con lo stesso numero di campi, se si desidera poter leggere a livello di record. Tieni presente che se scrivi un file sequenziale con record logici tali che l'ultimo blocco fisico contiene 254 caratteri, cioe' e' completo al momento della chiusura, viene occupato un settore in piu' solo in modo apparente; cioe' tale settore manca nel computo dei blocchi liberi e occupati. Se esegui il comando COLLECT, o VALIDATE, la directory torna in ordine e il file puo' essere letto senza problemi.

. campi di lunghezza fissa o variabile, tenendo presenti le limitazioni imposte dal comando INPUT#, che non puo' leggere piu' di 88 caratteri tra due CHR$(13) (RETURN). Inoltre va tenuto presente che, se il separatore (registrato) tra i campi e' la virgola, essi devono essere letti tutti da una sola istruzione INPUT#, che lavora sui dati compresi tra due CHR$(13). Una variabile che contiene solo spazi da' luogo a un campo nullo, cioe' due caratteri separatori vicini, che in fase di lettura sono riconosciuti come uno solo, e quindi viene perso un campo.

. in fase di scrittura i caratteri separatori tra i dati della lista agiscono come per la cassetta: il punto e virgola non aggiunge spazi, la virgola si.

. al momento della OPEN per scrivere viene creata la nuova entrata nella directory; essa viene completata al momento della CLOSE. Se il file non viene chiuso, l'entrata rimane in stato irregolare e il file non puo' essere utilizzato.

. non e' necessario aprire il canale 15, se non vuoi gestire gli errori tramite di esso, ma ti raccomandiamo di farlo. Infatti e' buona norma eseguire la routine di errore dopo ogni operazione disco.

Le istruzioni BASIC disponibili sono:

```
OPENlfn,dn,sa,"dr:nomef,tipo,modo"
```

per aprile il file. I parametri possono essere costanti e variabili:

. lfn, numero logico del file da 0 a 127.
. dn, 8 per la prima unita'.
. sa, canale per i dati da 2 a 14.
. dr, numero drive, 0 per default, puo' essere omesso.
. nomef, nome del file, massimo 16 caratteri.
. tipo, S, che significa sequenziale.
. modo, W per scrivere e R per leggere.

Puoi usare per la OPEN per scrivere questo formato:

```
OPENlfn,dn,sa,"@dr:nomef,S,W"
```

ottenendo di cancellare, se esiste, un file con lo stesso nome. Ti consigliamo di non usare questo formato, ma di cancellare il vecchio file con il comando SCRATCH e poi di aprirlo per scrivere.
Il DOS interpreta il comando e predispone il messaggio di errore in seguito all'esecuzione.
Possono essere aperti contemporaneamente fino a 3 file sequenziali su un floppy; il DOS li distingue in base al valore di "lfn".

```
PRINT#lfn,lista-dati
```

per scrivere i dati sul file aperto per scrivere, con numero logico "lfn". Non ripetiamo le considerazioni su lista-dati, gia' viste all'inizio.

```
INPUT#lfn,lista-dati
```

per leggere nelle variabili di lista-dati dal file aperto per leggere, con lo stesso valore di "lfn". Il tipo delle variabili deve concordare con il tipo dei dati che si leggono, altrimenti si ha errore.

```
GET#lfn,lista-dati
```

per leggere i dati carattere per carattere dal file aperto per leggere, con lo stesso valore di "lfn". E' meglio che lista-dati contenga solo variabili stringa.

```
CLOSElfn
```

per chiudere il file aperto con numero logico "lfn". La chiusura di un file aperto per scrivere fa aggiungere la segnalazione di EOF e provoca l'aggiornamento della directory e della BAM sul floppy. Per tutti i

tipi di file la CLOSE provoca l'aggiornamento in memoria della tabella di gestione dei file.

Come esempio abbiamo realizzato il programma SEQDISCO. Esso gestisce un archivio sequenziale di dati su disco, consentendo le seguenti operazioni:

.1) creazione ex-novo del file. I record devono essere forniti in ordine in base ai primi due campi, l'ordine viene controllato e non accetta record con i primi due campi uguali.

.2) lista completa del file su stampante, nell'ordine di registrazione dei record.

.3) aggiornamento del file, cioe':

. inserimento nuovi record, che devono essere forniti in ordine.

. modifica record esistenti, procedendo nell'ordine di registrazione dei record.

. cancellazione record, procedendo nell'ordine di registrazione dei record.

L'archivio viene mantenuto in ordine crescente in base ai primi due campi di ogni record, ma senza ricorrere a ordinamento; quindi i record devono essere caricati in questo ordine e il programma scarta quelli fuori ordine.
Il programma e' stato scritto ricorrendo alla tecnica dei sottoprogrammi; in conseguenza esso e' facilmente modificabile per adattarlo alle proprie esigenze.
Noi abbiamo lavorato con un record logico di lunghezza variabile, ma formato da un numero fisso di campi, 5 per ogni record, ognuno di lunghezza variabile. I campi sono separati tra loro dal carattere CHR$(13); questo ci consente di avere per ogni campo la massima lunghezza possibile, cioe' 88 caratteri. La gestione dell'archivio avviene a livello record, trattando sempre lo stesso numero di campi; in conseguenza e' necessario che ogni record abbia tutti i campi stabiliti. Per mantenere questa caratteristica, quando un campo manca esso viene registrato con il carattere CHR$(160), che corrisponde a SHIFT-spazio, e che in fase di lettura non da' luogo a un campo vuoto, come succederebbe con il carattere spazio normale.
Nella Figura 3.5 riportiamo uno schema a blocchi della fase di aggiornamento del file, che risulta la piu' complicata. Durante questa fase viene creato un file temporaneo, di nome TEMP, sul quale viene aggiornato il file vecchio; alla fine il file vecchio viene cancellato e al file TEMP viene cambiato il nome, assegnandogli quello del vecchio file.

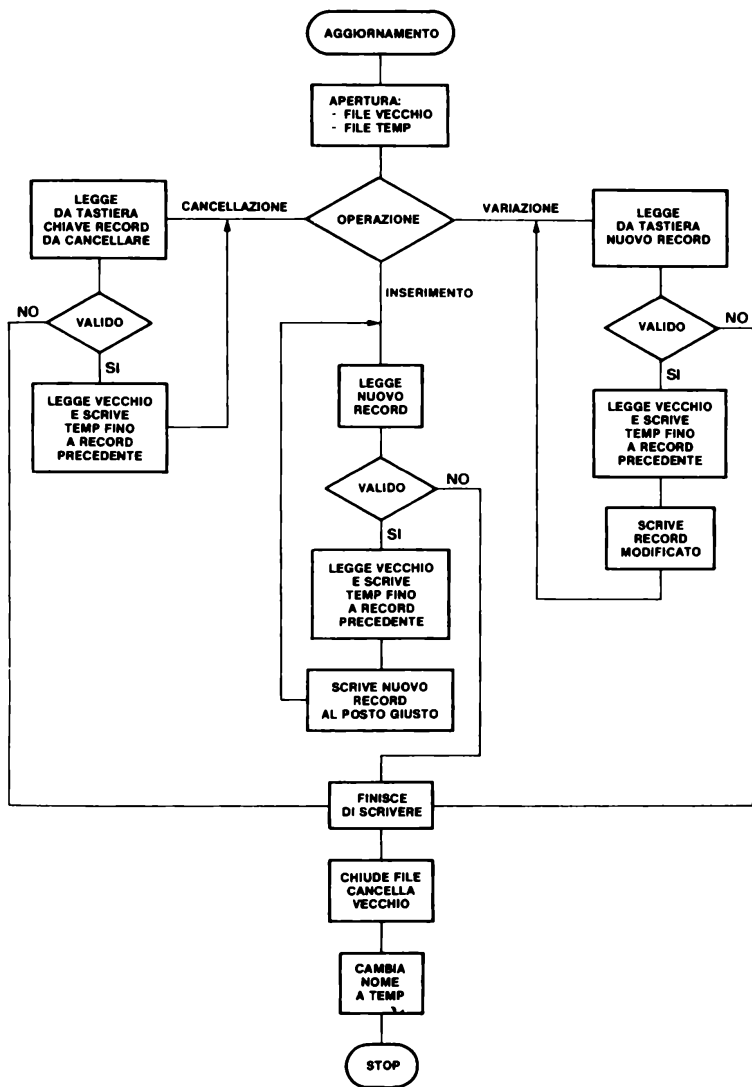



Figura 3.5 Diagramma a blocchi dell'aggiornamento

Il programma non presenta un menu' iniziale, ma pone successivamente domande sulle operazioni che si vogliono fare; a tali domande devi rispondere con S per SI e N per NO. Per uscire dalla richiesta di un

record devi rispondere con il carattere asterisco "*" al primo campo. Se manca un campo puoi rispondere solo con RETURN, provvede il programma a sostiuirlo con il carattere CHR$(160) (SHIFT-spazio). Ogni fase termina con lo STOP; se vuoi proseguire devi eseguire il comando RUN. All'inizio di ogni fase viene chiesto il nome del file. Il file puo' essere lungo a piacere, compatibilmente allo spazio disponibile sul floppy. Nella fase di lista i dati sono presentati ponendo su una riga i primi due campi e sull'altra gli altri 3, inoltre non viene usato un conta righe per il cambio del foglio.

```
1 REM SEQDISCO
3 REM CREAZIONE E GESTIONE FILE SEQUENZIALE
5 REM ***
7 REM PARAMETRI DI STAMPA-APERTURA CANALE 15
9 NF=4:PR=4:OPEN15,8,15:GOSUB397
11 REM ***
13 REM COSTANTI E VARIABILI
15 NC=5:CH$=CHR$(13):SP$="       "
17 DIMI$(NC),I1$(NC),D$(NC)
19 DATA "CAMPO1 ","CAMPO2 ","CAMPO3 "
21 DATA "CAMPO4 ","CAMPO5 "
23 FORK=1TONC:READD$(K):NEXTK
25 REM ***
27 REM SCELTA OPERAZIONE
29 PRINT"CREAZIONE FILE  S/N ";:INPUTR$
31 IFR$<>"S"THEN71
33 REM ***
35 REM CREAZIONE FILE
37 GOSUB297:GOSUB281:GOSUB303
39 PRINT#15,"I0":REM INIZIALIZZAZIONE
41 REM ***
43 REM APERTURA FILE PER SCRIVERE-CANALE 2
45 OPEN2,8,2,"0:"+NF$+",S,W":GOSUB397
47 LC$="":LN$="":REM VECCHI CAMPI 1 E 2
49 GOSUB233:REM RICHIESTA DATI
51 IFSW=0THEN61
53 CLOSE2:CLOSE15
55 PRINT"FINITO CARICAMENTO    FILE":STOP
57 REM ***
59 REM CONTROLLO ORDINE RECORD
61 IFI$(1)>LC$THEN67
63 IFI$(1)=LC$THENIFI$(2)>LN$THEN67
65 GOSUB289:GOTO49
67 LC$=I$(1):LN$=I$(2)
69 GOSUB267:GOTO49
71 REM ***
```

```
73 REM STAMPA FILE
75 PRINT"[[[STAMPA FILE S/N";:INPUTR$
77 IFR$<>"S"THEN103
79 GOSUB297:GOSUB281
81 PRINT#15,"I0":REM INIZIALIZZAZIONE
83 GOSUB303:GOSUB309
85 OPENNF,PR:PRINT#NF,"LISTA ARCHIVIO ";NF$
87 PRINT#NF
89 REM ***
91 REM LETTURA E STAMPA RECORD
93 GOSUB259:PRINT#NF,I$(1);SP$;I$(2)
95 FORK=3TONC:PRINT#NF,I$(K);SP$;:NEXTK
97 PRINT#NF:PRINT#NF:IFFS<>64THEN91
99 CLOSE3:CLOSENF:CLOSE15
101 PRINT"[FINITO LISTA":STOP
103 REM ***
105 REM AGGIORNAMENTO FILE
107 FF=0:PRINT"[AGGIORNAMENTO FILE"
109 GOSUB297:GOSUB281
111 PRINT#15,"I0":REM INIZIALIZZAZIONE
113 GOSUB303:GOSUB309
115 REM ***
117 REM APERTURA FILE TEMPORANEO PER SCRIVERE
119 OPEN2,8,2,"0:TEMP,S,W":GOSUB397
121 LC$="":LN$=""
123 PRINT"[[[TIPO DI VARIAZIONE:"
125 PRINT"1=INSERIMENTO":PRINT"2=VARIAZIONE"
127 PRINT"3=CANCELLAZIONE"
129 INPUTR:IFR<1ORR>3THEN129
131 FL=0:REM 1 SE ULT. REC. LETTO DA SCRIVERE
133 FF=0:REM 1 SE FINITO FILE INPUT
135 ONRGOTO137,185,209:REM SCELTA OPERAZIONE
137 REM ***
139 REM INSERIMENTO RECORD
141 GOSUB233:IFSW=0THEN155
143 GOSUB359:CLOSE3:CLOSE2
145 PRINT"[FINITO AGGIORNAMENTO"
147 REM ***
149 REM SISTEMAZIONE FILE SU DISCO
151 PRINT#15,"S0:"+NF$
153 PRINT#15,"R0:"+NF$+"=TEMP":CLOSE15:STOP
155 IFI$(1)>LC$THEN175:REM INSERISCE
157 REM ***
159 REM DATI IN DISORDINE
161 IFI$(1)<LC$THENGOSUB289:GOTO137
163 REM ***
165 REM PRIMO CAMPO UGUALE
167 IFI$(2)>LN$THEN175
169 IFI$(2)<LN$THENGOSUB289:GOTO137
171 GOSUB275:GOTO137
173 REM ***
175 REM DATI IN ORDINE
177 LC$=I$(1):LN$=I$(2):GOSUB317
179 IFI$(1)<>I1$(1)THEN183
181 IFI$(2)=I1$(2)THENGOSUB275:GOTO137
183 GOSUB267:GOTO137
185 REM ***
187 REM MODIFICA RECORD
189 GOSUB233:IFSW=0THEN193
191 GOTO143
193 IFI$(1)>LC$THEN199
195 IFI$(1)=LC$THENIFI$(2)>LN$THEN199
197 GOSUB289:GOTO185
199 LC$=I$(1):LN$=I$(2):GOSUB373
```

```
201 IFFL=1ANDFF=0THEN185
203 IFFL=1ANDFF=1THENFL=0:GOTO143
205 GOSUB267:IFFF=1THEN143
207 GOTO185
209 REM ***
211 REM CANCELLAZIONE RECORD
213 PRINTD$(1);:INPUTI$(1):IFI$(1)="*"THEN143
215 PRINTD$(2);:INPUTI$(2):IFI$(1)>LC$THEN221
217 IFI$(1)=LC$THENIFI$(2)>LN$THEN221
219 GOSUB289:GOTO209
221 LC$=I$(1):LN$=I$(2):GOSUB373
223 IFFF=1ANDFL=0THEN143
225 GOTO209
227 REM ***
229 REM SOTTOPROGRAMMI
231 REM ***
233 REM RICHIESTA DATI
235 SW=0:REM 1 SE FINITI DATI
237 PRINT"⌐ 1 ";D$(1);:INPUTI$(1)
239 IFI$(1)="*"THENSW=1:RETURN
241 FORK=2TONC:I$(K)=CHR$(160):NEXTK
243 FORK=2TONC:PRINTK;D$(K);:INPUTI$(K):NEXTK
245 PRINT"■CONFERMI S/N";:INPUTR$
247 IFR$="S"THENRETURN
249 PRINT"QUALE CAMPO ";:INPUT R
251 IFR<1ORR>NCTHENPRINT"⌐";:GOTO249
253 PRINTD$(R);:I$(R)=CHR$(160):INPUTI$(R)
255 PRINT"⌐";
257 FORK=1TONC:PRINTK;D$(K);I$(K):NEXTK:GOTO245
259 REM LETTURA RECORD DA DISCO
261 FORK=1TONC:INPUT#3,I$(K):FS=ST
263 GOSUB397:NEXTK:RETURN
265 REM ***
267 REM SCRITTURA NUOVO RECORD
269 FORK=1TONC:PRINT#2,I$(K);CH$;
271 GOSUB397:NEXTK:RETURN
273 REM ***
275 REM MESSAGGIO CAMPI 1 E 2 UGUALI
277 PRINT"DATI CAMPI 1 E 2 UGUALI":GOSUB281:RETURN
279 REM ***
281 REM ATTESA TASTO
283 GETA$:IFA$=""THEN281
285 RETURN
287 REM ***
289 REM MESSAGGIO FUORI ORDINE
291 PRINT"■DATI FUORI ORDINE":GOSUB281
293 RETURN
295 REM ***
297 REM RICHIESTA DISCO DATI
299 PRINT"■MONTA DISCO DATI":RETURN
301 REM ***
303 REM RICHIESTA NOME FILE
305 INPUT"NOME FILE ";NF$:RETURN
307 REM ***
309 REM APERTURA FILE PER LEGGERE-CANALE 3
311 OPEN3,8,3,"0:"+NF$+",S,R":GOSUB397
313 RETURN
315 REM ***
317 REM LEGGE RECORD E SCRIVE FINO AL
319 REM RECORD PRECEDENTE SU TEMP
321 IFFF=1ANDFL=0THENRETURN
323 IFFL<>0THEN329
325 GOSUB343
327 IFFS=64THENFF=1
```

```
329 IFI1$(1)<I$(1)THEN335
331 IFI1$(1)=I$(1)THENIFI1$(2)<I$(2)THEN335
333 FL=1:RETURN
335 GOSUB351:FL=0
337 IFFF=1THENRETURN
339 GOTO325
341 REM ***
343 REM LETTURA RECORD
345 FORK=1TONC:INPUT#3,I1$(K):FS=ST
347 GOSUB397:NEXTK:RETURN
349 REM ***
351 REM SCRITTURA RECORD
353 FORK=1TONC:PRINT#2,I1$(K)
355 GOSUB397:NEXTK:RETURN
357 REM ***
359 REM SCRIVE FILE TEMP FINO ALLA FINE
361 REM DEL FILE DI INPUT
363 IFFF=1ANDFL=0THENRETURN
365 IFFL<>0THENFL=0:GOSUB351:GOTO359
367 GOSUB343:GOSUB351:IFFS=64THENFF=1:RETURN
369 GOTO367
371 REM ***
373 REM COPIA FILE FINO AL RECORD CERCATO
375 IFFF=1ANDFL=0THEN388
377 IFFL<>0THEN381
379 GOSUB343:IFFS=64THENFF=1
381 IFI1$(1)<I$(1)THEN389
383 IFI1$(1)=I$(1)THENIFI1$(2)<I$(2)THEN389
385 IFI1$(1)=I$(1)ANDI1$(2)=I$(2)THENFL=0:RETURN
387 FL=1
388 PRINT"NON TROVATO":GOSUB281:RETURN
389 GOSUB351
391 IFFF=1THEN388
393 GOTO379
395 REM ***
397 REM ROUTINE ERRORE
399 INPUT#15,EN,EM$,ET,ES
401 IFEN=0THEN RETURN
403 PRINT"ERRORE DISCO"
405 PRINTEN,EM$,ET,ES:STOP
```

ELENCO VARIABILI

.NF, 4, numero logico file di stampa.
.PR, 4, dn della periferica di stampa.
.NC, 5, numero campi del record.
.CH$, CHR$(13), carattere separatore RETURN.
.SP$, 4 spazi.
.I$(NC), vettore per leggere i dati dalla tastiera.
.I1$(NC), vettore per leggere da disco il record.
.D$(NC), vettore per le descrizioni dei campi, contenute nelle linee DATA.
.R$, variabile per risposte.
.LC$, primo campo vecchio.
.LN$, secondo campo vecchio.
.FS, memorizzazione parola di stato ST.
.EN, EM$, ET, ES, variabili per controllo errori.

.SW, switch per controllare l'ingresso dei dati: 0 per si dati, 1 per finiti i dati.
.FF, switch per fine file di input, 1 per finito.
.FL, switch per utilizzo ultimo record letto: 0 utilizzato, 1 non utilizzato.
.R, K, variabili per il controllo dei cicli.

Per adattare il programma alle tue esigenze puoi fare le seguenti modifiche:
.PR=3 per listare sul video.
.NC per passare da 5 al numero di campi desiderato.
.linee DATA 19/21, per modificare le diciture dei campi.
.linee da 85 a 103 per ottenere un diverso formato di stampa.

COMMENTO A SEQDISCO
.1/23: definizione costanti e variabili, apertura canale 15, che viene chiuso alla fine di ogni fase.
.25/55: creazione file ex-novo.
.57/69: controllo ordine record nella fase di creazione ex-novo.
.71/103: stampa file.
.105/121: inizio fase di aggiornamento, apertura file temporaneo.
.123/135: scelta tipo aggiornamento, azzeramento switch.
.137/145: fase inserimento nuovi record.
.147/155: sistemazione file temporaneo per fine aggiornamento.
.157/171: controllo ordine dei record e uguaglianza primi 2 campi nella fase di inserimento.
.173/183: inserimento se dati in ordine.
.185/207: fase aggiornamento record per modifiche.
.209/225: fase cancellazione record.
.227/263: inizio sezione per i sottoprogrammi. Richiesta dati da tastiera.
.265/271: scrittura nuovo record.
.273/277: messaggio per campi chiave uguali.
.279/285: attesa tasto per proseguire.
.287/293: messaggio record fuori ordine.
.295/299: richiesta disco dati.
.301/305: richiesta nome file.
.307/313: apertura file per leggere.
.315/339: avanzamento e copiatura file fino al record precedente, per la fase di modifica record.
.341/347: lettura record da disco.
.349/355: scrittura record su TEMP.

.357/369: finisce di ricopiare file su TEMP.
.371/393: copia file fino al record cercato, per la fase di cancellazione.
.395/405: routine di controllo errore disco.

3.6 FILE RANDOM DI DATI

Questa denominazione di file e' abbastanza impropria, ma ormai in uso; per questa ragione anche noi chiamiamo random questi file, che in realta' sono ad accesso diretto. Essi sono gestiti con il gruppo dei comandi DOS per l'accesso diretto, che non comportano una registrazione nella directory del disco. La cosa non e' consigliabile, dal momento che su un floppy che contenga file random non si possono eseguire i comandi VALIDATE e COLLECT che controllano e sistemano la BAM e la directory. Noi per completare l'argomento presentiamo un programma che gestisce in modo RANDOM un archivio di dati, in modo pericoloso, cioe' senza registrazione nella BAM; e un secondo programma che gestisce un file in modo RANDOM-USER, con l'accorgimento di preparare preventivamente il floppy in modo che il file sia anche registrato nella directory. In questo secondo caso sono eseguibili sul floppy anche i comandi VALIDATE e COLLECT.
Nell'organizzazione diretta il file viene gestito a livello di record fisico, cioe' di settore, conoscendone l'indirizzo t,s.
Il record logico deve essere definito con tutti i suoi campi e adattato alla struttura fisica. Senza addentrarsi in tecniche di programmazione piuttosto sofisticate, il modo migliore per gestire questo tipo di file e' di definire un record logico di lunghezza fissa, formato da un numero fisso di campi di lunghezza fissa, in modo tale che risulti di lunghezza multipla o sottomultipla di un settore, considerato di 254 caratteri. Preferiamo usare solo 254 caratteri in un settore per poter mantenere l'uso dei primi 2 byte per il concatenamento tra i settori, come per gli altri tipi di file.
Dobbiamo ora affrontare un altro argomento relativo all'organizzazione dei file, cioe' la possibilita' di reperire rapidamente il record voluto. Nell'esempio relativo ai file sequenziali abbiamo preparato il file mantenendo i record in ordine in base a due campi chiave, pero' la ricerca poteva avvenire solo leggen-

do i record in sequenza. In questo caso possiamo accedere al record desiderato, se conosciamo l'indirizzo t,s. Come facciamo a sapere, per esempio in un archivio relativo a una biblioteca, che il libro ARITMETICA PRATICA dell'autore PIPPO si trova in un determinato settore? Potremmo avere una lista su carta che ci da' queste informazioni, cioe' un indice da consultare manualmente prima di una ricerca. Ma, dal momento che abbiamo un calcolatore, e' meglio organizzarci per fargli svolgere tutto il lavoro in modo automatico.
In conseguenza quando creiamo un file ad accesso diretto, fisico per i file random, logico per i file relativi che trattiamo nel prossimo paragrafo, dobbiamo creare anche un indice sequenziale, organizzato in base a uno o piu' campi di ordinamento, che ci fornisca gli indirizzi, fisici o logici, necessari per reperire un record.
Il file INDICE, che e' un normale file sequenziale, deve essere abbastanza corto se possibile. L'altro file, quello diretto, si chiama PRINCIPALE. La situazione ideale e' che sia possibile caricare in memoria il file indice prima di iniziare l'elaborazione del file principale; in tale modo le operazioni di ricerca nell'indice risultano molto veloci. Se questo non e' possibile, il file indice puo' venir letto a blocchi, andando a leggere i blocchi successivi quando servono. Comunque la lettura del file indice risulta piu' veloce di quanto sarebbe il trattamento di un file principale, con tanti campi, di tipo sequenziale.

Nel Paragrafo 3.2, nella sezione GESTIONE DIRETTA, sono riportati i comandi per il DOS. Il trattamento dei file random impegna 2 canali, il canale 15, per i comandi, e un canale, da 2 a 14 per i dati. Nel buffer dei dati agiscono le normali istruzioni PRINT#, INPUT# e GET#.
Nel seguito riportiamo due esempi di programmi per gestire un archivio random normale e uno organizzato in modo da proteggere le registrazioni. Nei commenti trattiamo completamente l'argomento.
Ti facciamo notare che i file random, dopo l'apertura, possono essere letti e scritti, mescolando le due operazioni.

ARCHIVIO RANDOM

Abbiamo preparato il programma ARCHIRANDOM per gestire un file RANDOM con indice. Le funzioni del program-

ma sono le seguenti:

.1) crea sul floppy un archivio RANDOM, che non ha un nome registrato nella directory, e il file indice INDI1, usando come campi chiave i primi due campi del record logico, e ordinando l'indice in ordine crescente in base ai campi chiave.

.2) lista tutto l'archivio in ordine secondo l'indice INDI1.

.3) aggiorna l'archivio operando:

.3a) aggiunta di nuovi record,

.3b) modifica di record esistenti,

.3c) cancellazione di record.

Gli aggiornamenti avvengono sul file principale e sull'indice INDI1.

.4) crea un indice secondario INDI2 in base a un qualunque campo del record, escluso il primo, in ordine decrescente del campo chiave.

.5) lista tutto l'archivio in ordine secondo INDI2.

Per gestire veramente un archivio di dati sono necessarie anche altre funzioni, come ricerca e stampa di un record o di un gruppo di record, estrazione di record con determinate caratteristiche, ecc.. Partendo dal nostro esempio, non ti sara' difficile sviluppare in modo simile altre parti del programma. Il nostro scopo e' di mostrarti la problematica della gestione degli archivi di dati e alcuni esempi completi dell'uso delle istruzioni del DOS per la gestione diretta dei file.

Il programma e' stato sviluppato con la tecnica dei sottoprogrammi e in modo che, con poche modifiche, puo' essere adattato ad altre situazioni.
Abbiamo organizzato un record logico formato da 14 campi di lunghezza fissa, realizzando un archivio anagrafico. Elenchiamo nell'ordine di registrazione le descrizioni dei campi e la loro lunghezza in caratteri:

1)	cognome	20	8)	data nascita	8
2)	nome	15	9)	titolo studio	20
3)	indirizzo	30	10)	occup. attuale	20
4)	CAP/citta'	25	11)	occup. preced.	20
5)	provincia	11	12)	stato civile	1
6)	telefono	10	13)	nota 1	20
7)	luogo nascita	20	14)	nota 2	20

alla lunghezza di ogni campo devi aggiungere un carat-

tere di fine campo, che per noi e' CHR$(13) (RETURN). Avremmo potuto usare anche CHR$(44) (virgola), ma con il carattere RETURN siamo piu' liberi di leggere con ogni INPUT# i campi desiderati. La somma delle lunghezze dei campi piu' il carattere separatore da' 254, cioe' un settore non usando i primi due caratteri. Abbiamo quindi coincidenza tra il record logico e il record fisico. I dati sono registrati nel settore partendo dal terzo carattere, cioe' posizionando il puntatore interno a 2.
Per il file indice INDI1 ogni record e' formato dai seguenti campi:

. chiave: cognome+nome, di 35 caratteri, completando con spazi ogni campo se piu' corto della sua lunghezza,

. indirizzo traccia, t, numero intero,

. indirizzo settore, s, numero intero.

Per il file INDI2, che viene creato quando si esegue il passo 4, abbiamo 3 campi per record, come per INDI1, ma le dimensioni del primo campo dipendono dal campo scelto come chiave. Dal momento che un campo chiave puo' anche essere vuoto, se esso ha lunghezza 0, viene sostituito in INDI2 dal carattere CHR$(160). In conseguenza nell'ordinamento decrescente di INDI2 i campi chiave vuoti risultano i primi.

Dobbiamo fare alcune considerazioni sull'occupazione di memoria. Infatti l'indice INDI1 viene creato e gestito in memoria e poi scritto su disco, quindi richiede spazio per i 3 vettori nei quali viene memorizzato. Non ci sembra utile fare dei conti precisi per il nostro archivio, dal momento che tu probabilmente personalizzerai il programma secondo le tue esigenze. Per fare i conti, dopo aver caricato in memoria il programma, devi procedere cosi':

.leggere il puntatore a inizio variabili, nei byte 45 e 46, e calcolarne il valore;

.leggere il puntatore di fine zona BASIC, nei byte 55 e 56, e calcolarne il valore;

.calcolare la differenza tra il secondo e il primo valore, che rappresenta il numero di byte a disposizione per tutte le variabili (tale numero lo ottieni anche eseguendo CLR:PRINT FRE(0));

.calcolare approssimativamente l'occupazione di memoria per ogni record di INDI1 e quindi ricavare il numero di elementi che puo' avere ognuno dei 3 vettori. A questo punto hai fatto i conti con la memoria del calcolatore.

Per il dischetto devi tener presente che ogni record del file random occupa un settore, calcolare l'occupazione di ogni record di INDI1 e di INDI2 in modo approssimato, dedurre il numero di record registrabili e controllare se tale numero si accorda con il calcolo fatto per la memoria del calcolatore.
E' importante fare bene questi conti per non avere spiacevoli sorprese. In caso un archivio puo' essere suddiviso in diverse parti e tenuto su piu' floppy.

Il programma lavora usando un dischetto apposito per i dati; su questo nel settore 0 della traccia 1 sono memorizzati i dati generali del file, e precisamente:

 . data di aggiornamento,
 . numero dei record presenti,
 . indirizzo del blocco, traccia e settore, dell'ultimo record registrato.

I record del file random sono registrati a partire dalla traccia 1, settore 1.
All'inizio viene presentato un menu' di scelta della funzione. Il programma procede con domande semplici nelle diverse situazioni. Per uscire dalla richiesta dati devi rispondere con il carattere dollaro "$" al cognome. Quando compare un messaggio senza richiesta di risposta per proseguire devi premere un tasto.
Per uscire dal menu' senza operare devi scegliere la funzione 9 (FINE).

```
1 REM ARCHIRANDOM
3 REM ***********
5 REM DEFINIZIONE COSTANTI E VARIABILI
7 REM **********************************
9 NC=14:REM NUMERO CAMPI RECORD
11 S1$="DATA ULTIMO AGG.      "
13 S2$="FINITO SPAZIO ASSEGNATO"
15 SP$="":FORK=1TO79:SP$=SP$+" ":NEXTK
17 CH$=CHR$(13):LIM$=CHR$(99)+CHR$(99)+CHR$(99)
19 DIMD$(NC):DIMY$(NC):DIML(NC)
21 DATA"COGNOME:","NOME    :","INDIR.  :"
23 DATA"CITTA'  :","PROV.   :","TELEF.  :"
25 DATA"L.NASC.:","D.NASC.:","T.STUD.:"
27 DATA"OCC.AT.:","OCC.PR.:","ST.CIV.:"
29 DATA"NOTA 1  :","NOTA 2  :"
31 DATA20,15,30,25,11,10,20,8,20,20,20,1,20,20
33 FORK=1TONC:READD$(K):NEXTK
35 FORK=1TONC:READL(K):NEXTK
37 REM PRESENTAZIONE MENU' E SCELTE
39 REM ******************************
41 OPEN15,8,15
43 PRINT"♥QQ";TAB(10);"GESTIONE ARCHIVIO":PRINT
```

```
45 PRINTTAB(10)"1=INIZIO EX-NOVO"
47 PRINTTAB(10)"2=AGGIORNAMENTO"
49 PRINTTAB(10)"3=LISTA PRINCIPALE"
51 PRINTTAB(10)"4=CREAZ.IND.SEC."
53 PRINTTAB(10)"5=LISTA SECONDARIA"
55 PRINTTAB(10)"9=FINE"
57 INPUT"COSA SCEGLI ";X
59 IFX<1ORX>5ANDX<>9THENPRINT"]";:GOTO57
61 PRINT:GOSUB225
63 R$="":PRINTTAB(10)"MONTA DISCO DATI"
65 GETR$:IFR$=""THEN65
67 IFX=1THEN77
69 PRINT#15,"I"
71 GOSUB185:N=K:PRINT"PRESENTI ";K;" RECORD"
73 PRINTS1$;G1$;"/";M1$;"/";A1$
75 IFX<>2THEN83
77 REM INIZIO EX-NOVO ARCHIVIO
79 REM ***********************
81 PRINT"      QUANTI RECORD ";:INPUT N
83 DIMC$(N),T%(N),S%(N)
85 IFX<>1THENGOSUB201
87 IFX=9THEN507
89 ONXGOTO251,303,435,467,489
91 STOP
93 REM INGRESSO DATI
95 REM *************
97 PRINT"]INGRESSO DATI"
99 FORJ=1TONC:Y$(J)="":NEXTJ
101 PRINT"PER USCIRE $ PER COGNOME";
103 PRINT"▮SE MANCANO DATIPREMI SOLO RETURN"
105 PRINTD$(1);:INPUTY$(1)
107 IFY$(1)="$"THENW=1:RETURN
109 FORJ=2TONC:PRINTD$(J);:INPUTY$(J):NEXTJ
111 REM SISTEMA LUNGHEZZA DATI
113 REM **********************
115 FORJ=1TONC:Y$(J)=LEFT$(Y$(J)+SP$,L(J))
117 NEXTJ:GOSUB379:RETURN
119 REM SCRITTURA INDICE PRINCIPALE
121 REM ***************************
123 PRINT#15,"S0:INDI1":PRINT#15,"I"
125 OPEN10,8,10,"INDI1,S,W":GOSUB213
127 FORJ=1TOK
129 PRINT#10,C$(J);CH$;T%(J);CH$;S%(J);CH$;
131 GOSUB213:NEXTJ
133 CLOSE10:RETURN
135 REM SCRITTURA NEL BUFFER
137 REM ********************
139 FORJ=1TONC:Y$(J)=LEFT$(Y$(J)+SP$,L(J))
141 PRINT#11,Y$(J);CH$;:GOSUB213
143 NEXTJ:RETURN
145 REM ALLOCA TRACCIA E SETTORE
147 REM ************************
149 PRINT#15,"B-A:"0;T;S
151 INPUT#15,EN,EM$,ET,ES
153 IFEN=0THENRETURN
155 IFEN<>65THEN221
157 IFET=18THENT=19:S=0:GOTO149
159 T=ET:S=ES:GOTO149
161 REM PUNTATORE NEL BUFFER
163 REM ********************
165 PRINT#15,"B-P:"11;2:GOSUB213:RETURN
167 REM SCRITTURA RECORD
169 REM ****************
171 PRINT#15,"U2:"11;0;T;S:GOSUB213:RETURN
```

```
173 REM LETTURA RECORD
175 REM **************
177 PRINT#15,"I":OPEN11,8,11,"#":GOSUB213
179 PRINT#15,"U1:"11;0;T;S:GOSUB213:GOSUB165
181 FORJ=1TONC:INPUT#11,Y$(J):GOSUB213
183 NEXTJ:CLOSE11:RETURN
185 REM LETT. DATI DISCO
187 REM ****************
189 OPEN11,8,11,"#":GOSUB213
191 PRINT#15,"U1:"11;0;1;0:GOSUB213
193 GOSUB161:INPUT#11,R$,K,T,S:GOSUB213
195 G1$=LEFT$(R$,2):M1$=MID$(R$,3,2)
197 A1$=RIGHT$(R$,2):CLOSE11
199 TT=T:SS=S:RETURN
201 REM LETT. INDICE
203 REM ************
205 OPEN10,8,10,"INDI1,S,R":GOSUB213
207 FORJ=1TOK
209 INPUT#10,C$(J),T%(J),S%(J):GOSUB213
211 NEXTJ:CLOSE10:RETURN
213 REM ROUTINE ERRORE
215 REM **************
217 INPUT#15,EN,EM$,ET,ES
219 IFEN=0THENRETURN
221 PRINT"█ERRORE DISCO"
223 PRINTEN,EM$,ET,ES:CLOSE15:STOP
225 REM DATA DISCO
227 REM **********
229 PRINT"DATA PER DISCO"
231 INPUT"  GG,MM,AA████████████";G$,M$,A$:RETURN
233 REM SCRITT. IND. SEC.
235 REM *****************
237 PRINT#15,"S:INDI2":PRINT#15,"I"
239 OPEN10,8,10,"INDI2,S,W"
241 FORJ=1TOK
243 PRINT#10,C$(J);CH$;T%(J);CH$;S%(J);CH$;
245 GOSUB213:NEXTJ:CLOSE10:RETURN
247 GETR$:IFR$=""THEN247
249 RETURN
251 REM INIZIALIZZAZIONE DISCO
253 REM **********************
255 PRINT"NOME DISCO";:INPUTN$:T=1:S=0
257 REM DATI DISCO SETTORE 1,0
259 REM **********************
261 PRINT#15,"N0:"+N$+",99"
263 CLOSE15:OPEN15,8,15:PRINT#15,"I"
265 REM DATA AGG. E NUM. RECORD
267 REM ***********************
269 OPEN11,8,11,"#":GOSUB213:GOSUB149:GOSUB165
271 K=0
273 PRINT#11,G$M$A$CH$;K;CH$;1;CH$;1;CH$;
275 REM PRIMO SETTORE DATI 1,1
277 REM **********************
279 GOSUB213
281 PRINT#15,"U2:"11;0;T;S:GOSUB213
283 K=1:W=0:T=1:S=1
285 GOSUB93
287 IFW=1THENK=K-1:CLOSE11:GOTO507
289 GOSUB145:GOSUB161:GOSUB135
291 REM AGGIORNAMENTO INDICE
293 REM ********************
295 C$(K)=Y$(1)+Y$(2):T%(K)=T:S%(K)=S
297 GOSUB167:K=K+1
299 IFK>NTHENK=K-1:PRINTS2$:CLOSE11:GOTO507
```

```
301 GOTO285
303 REM AGGIORNAMENTO
305 REM *************
307 PRINTTAB(10);"■■■AGGIORNAMENTO ARCHIVIO"
309 PRINT:PRINTTAB(10);"1=CORREZIONE"
311 PRINTTAB(10);"2=AGGIUNTA ELEM."
313 PRINTTAB(10);"3=CANCELL.ELEM."
315 PRINTTAB(10);"9=FINE"
317 INPUT"COSA SCEGLI ";X:IFX=9THEN507
319 IFX<1ORX>3THEN307
321 IFX=2THEN401
323 IFX=3THEN417
325 GOTO359
327 REM RICERCA RECORD
329 REM **************
331 PRINT"■";D$(1);:Y$(1)="":INPUTY$(1):W=0
333 IFY$(1)="$"THENW=1:RETURN
335 PRINTD$(2);:Y$(2)="":INPUTY$(2)
337 Y$(1)=LEFT$(Y$(1)+SP$,L(1))
339 Y$(2)=LEFT$(Y$(2)+SP$,L(2))
341 INPUT"QUALE OCCORRENZA ";X
343 IFX<=0THENPRINT"■";:GOTO341
345 FORJ=1TOK:IFC$(J)=Y$(1)+Y$(2)THEN353
347 NEXTJ
349 J=J-1:PRINT"NON TROVATO ";Y$(1);"  ";Y$(2)
351 GOSUB247:GOTO327
353 IFX<>1THENX=X-1:GOTO347
355 T=T%(J):S=S%(J)
357 I=J:J=K:NEXTJ:RETURN
359 REM CORREZIONE
361 REM **********
363 GOSUB327:IFW=1THEN307
365 GOSUB173
367 REM VA A MODIFICA DATI
369 REM ******************
371 GOSUB379
373 PRINT#15,"I":OPEN11,8,11,"#":GOSUB161:GOSUB135
375 C$(I)=Y$(1)+Y$(2):T%(I)=T:S%(I)=S
377 GOSUB167:CLOSE11:GOTO325
379 REM CONTROLLO DATI E MODIFICHE
381 REM **************************
383 PRINT"■CONTROLLO DATI E MODIFICHE"
385 FORJ=1TONC:PRINTJ;" ";D$(J);" ";Y$(J):NEXTJ
387 INPUT"■TUTTO BENE S/N ";X$:IFX$="S"THENRETURN
389 PRINT"QUALE CAMPO (0 USCITA)";:INPUTX
391 IFX=0THENRETURN
393 IFX>NCORX<1THENPRINT"■■":GOTO387
395 Y$(X)=""
397 INPUTY$(X):Y$(X)=LEFT$(Y$(X)+SP$,L(X))
399 GOTO383
401 REM AGGIUNTA
403 REM ********
405 OPEN11,8,11,"#":GOSUB213
407 PRINT"■AGGIUNTA NUOVI RECORD"
409 PRINT"PRESENTI ";K;" RECORD"
411 PRINT"PUOI AGGIUNGERE ";N-K;" RECORD"
413 GOSUB247
415 T=TT:S=SS:W=0:K=K+1:GOTO285
417 REM CANCELLAZIONE
419 REM *************
421 M=0
423 GOSUB327:IFW=1THEN431
425 X=L(1)+L(2):C$(I)=LEFT$(LIM$+SP$+SP$,X)
427 M=M+1
```

```
429 PRINT#15,"B-F:"0;T;S:GOSUB213:GOTO423
431 REM SIST. INDICE
432 REM ************
433 GOSUB533:K=K-M:GOTO517
435 REM LISTA FILE
437 REM **********
439 T$="LISTA PER INDICE PRIM."
441 PRINT"ACCENDI STAMPANTE      PREMI UN TASTO"
443 GETR$:IFR$=""THEN443
445 PRINT"□";T$:OPEN4,4
447 PRINT#4:PRINT#4:PRINT#4,T$
449 FORJ=1TO10:PRINT#4:NEXTJ
451 L=2:FORI=1TOK:T=T%(I):S=S%(I):GOSUB173
453 PRINT#4,Y$(1);"  ";Y$(2)
455 PRINT#4,Y$(3);"  ";Y$(4);"  ";Y$(5)
457 FORM=6TONC:PRINT#4,D$(M);Y$(M):NEXTM
459 PRINT#4:PRINT#4
461 IFL=5THENPRINT#4:L=0
463 L=L+1:NEXTI
465 CLOSE4:GOTO527
467 REM CREAZIONE INDICE SECONDARIO
469 REM ***************************
471 PRINT"□INDICE SECONDARIO"
473 INPUT"■QUALE CAMPO ";X$
475 X=VAL(X$):IFX<2ORX>NCTHENGOTO473
477 GOSUB119
479 FORI=1TOK:T=T%(I):S=S%(I):GOSUB173
480 IFLEN(Y$(X))=0THENY$(X)=CHR$(160)
481 C$(I)=Y$(X):NEXTI
483 GOSUB233:GOSUB201
485 PRINT"FINITO IND. SEC."
487 GOTO527
489 REM LISTA PER INDICE SECONDARIO
491 REM ***************************
493 PRINT"□LISTA IND. SEC."
495 OPEN10,8,10,"INDI2,S,R":GOSUB213
497 FORJ=1TOK:INPUT#10,C$(J),T%(J),S%(J)
499 GOSUB213:NEXTJ
501 CLOSE10:GOSUB561:GOSUB233
503 T$="LISTA IND. SEC. "
505 GOTO441
507 REM CHIUSURA
509 REM ********
511 PRINT"CHIUSURA ARCHIVIO"
513 PRINT"IND. PRINC. DA ORDIN. S/N ":INPUTR$
515 IFR$="S"THENGOSUB533
517 GOSUB119:OPEN11,8,11,"#":GOSUB213
519 PRINT#15,"B-P:"11,2
521 PRINT#11,G$M$A$CH$;K;CH$;T;CH$;S;CH$;
523 GOSUB213
525 PRINT#15,"U2:"11;0;1;0:GOSUB213:CLOSE11
527 PRINT"FINITO AGGIORNAMENTO"
529 PRINT"SONO PRESENTI ";K;" RECORD"
531 CLOSE15:STOP
533 REM ORDINAMENTO CRESCENTE
535 REM *********************
537 L=K-1
539 W=0
541 FORJ=1TOL
543 IFC$(J)<=C$(J+1)THEN553
545 R$=C$(J):C$(J)=C$(J+1):C$(J+1)=R$
547 X=T%(J):T%(J)=T%(J+1):T%(J+1)=X
549 X=S%(J):S%(J)=S%(J+1):S%(J+1)=X
551 W=1
```

```
553 NEXTJ
555 IFW=0THENRETURN
557 IFL=1THENRETURN
559 L=L-1:GOTO539
561 REM ORDINAMENTO DECRESCENTE
563 REM **********************
565 L=K-1
567 W=0
569 FORJ=1TOL
571 IFC$(J)>=C$(J+1)THEN581
573 R$=C$(J):C$(J)=C$(J+1):C$(J+1)=R$
575 X=T%(J):T%(J)=T%(J+1):T%(J+1)=X
577 X=S%(J):S%(J)=S%(J+1):S%(J+1)=X
579 W=1
581 NEXTJ
583 IFW=0THENRETURN
585 IFL=1THENRETURN
587 L=L-1:GOTO567
```

ELENCO VARIABILI

.NC, numero campi del record logico, 14 per noi.

.S1$,S2$,T$, stringhe descrittive.

.SP$, 79 spazi, tale stringa deve contenere un numero di spazi pari al campo piu' lungo del record, infatti serve per aggiungere spazi ai campi piu' corti o vuoti. Noi abbiamo usato 79, per la compatibilita' con altri calcolatori COMMODORE, ma potrebbe essere anche 88.

.CH$, CHR$(13), per separare i campi con RETURN.

.LIM$, tre caratteri CHR$(99), serve per il campo chiave dei record cancellati.

.D$(NC), descrizioni dei campi del record, da prelevare dalle frasi DATA presenti nelle linee da 21 a 29.

.Y$(NC), campi dati.

.L(NC), lunghezze dei campi dati, da prelevare dalla linea DATA 31.

.K,J,L, variabili di controllo.

.X,R$,X$, variabili per risposte.

.N, numero record.

.G1$,M1$,A1$, data precedente.

.G$,M$,A$, data aggiornamento.

.C$(N), chiavi dell'indice.

.T%(N),S%(N), tracce e settori dell'indice.

.W, switch per lettura dati, 1 se finiti i dati.

.T,S, traccia e settore.

.EN,EM$,ET,ES, variabili per messaggio errore.

.N$, nome disco e archivio.

.TT,SS, indirizzo settore precedente.

.M, numero record cancellati.

Le modifiche per adattare il programma alle tue esigenze riguardano:
.il numero dei campi NC,
.le frasi DATA delle descrizioni e delle lunghezze dei campi.
.la parte di programma che lista i record.
Risulta piu' difficile modificare il fatto che l'indice e' sui primi due campi; puoi ovviare all'inconveniente dimensionando uno dei due campi a lunghezza 1 e lasciandolo vuoto.
Le due routine di ordinamento risultano un po' lente; potrebbero essere sostituite con routine di ordinamento in linguaggio macchina. Esse lavorano sui 3 vettori che contengono l'indice, o principale o secondario.
Il programma accetta record con i primi due campi uguali; in conseguenza nella ricerca chiede anche l'occorrenza del record.

COMMENTO A ARCHIRANDOM
.1/35: definizione costanti e variabili.
.37/91: presentazione menu' e scelte.
.93/117: sottoprogramma ingresso dati e sistemazione lunghezza campi.
.119/133: sottoprogramma scrittura INDI1 su floppy.
.135/143: sottoprogramma scrittura nel buffer.
.145/159: sottoprogramma allocazione traccia e settore.
.161/165: sottoprogramma posizionamento puntatore.
.167/171: sottoprogramma scrittura settore sul floppy.
.173/183: sottoprogramma lettura record.
.185/199: sottoprogramma lettura dati generali disco.
.201/211: sottoprogramma lettura INDI1 in memoria.
.213/223: sottoprogramma routine errore disco.
.225/231: sottoprogramma richiesta data aggiornamento.
.233/249: sottoprogramma scrittura indice INDI2.
.251/301: fase 1 del menu', inizio ex-novo archivio e caricamento record.
.303/433: fase 2 del menu', aggiornamento: modifica, aggiunta e cancellazione. Quando si cancella un record viene liberato il settore occupato.
.435/465: fase 3 del menu', lista archivio.
.467/487: fase 4 del menu', creazione INDI2.
.489/531: fase 5 del menu', lista per indice secondario.
.533/559: sottoprogramma ordinamento crescente indi-

ce, con eliminazione dei record cancellati.
.561/587: sottoprogramma ordinamento decrescente indice.

ARCHIVIO RANDOM-USER

Per organizzare un archivio di questo tipo e' necessario eseguire preventivamente un programma che impegni su disco il numero di settori che servono per il file random, creando un file sequenziale di record "dummy" (vuoti), al quale, per distinguerlo dagli altri, possiamo assegnare il tipo USR. Impegnando i settori in questo modo essi vengono concatenati sui primi 2 caratteri e il file di tipo USR viene registrato nella directory. Dobbiamo conservare gli indirizzi di traccia e settore utilizzati dal file USR, e, quando scriviamo i record del file random, dobbiamo andare a scrivere sugli stessi settori, senza eseguire il comando B-A di allocazione. In questo modo le nostre registrazioni non vengono danneggiate dai comandi COLLECT e VALIDATE.

Abbiamo voluto migliorare ulteriormente l'organizzazione del nostro programma, in modo che esso possa essere utilizzato per generare archivi di tipo diverso. Abbiamo stabilito di mantenere la lunghezza del record logico al massimo a 254 caratteri, utilizzando un settore per ogni record logico. Inoltre abbiamo mantenuto le caratteristiche funzionali del programma ARCHIRANDOM.
Per realizzare questo archivio dobbiamo:
.preparare un programma, di nome STRUTTFILE, che mediante un colloquio video/tastiera, ci consente di definire la struttura del record logico dell'archivio, fissando il numero dei campi, il loro nome e la loro lunghezza, il nome del disco e dell'archivio, la data iniziale. STRUTTFILE controlla che la lunghezza dei campi non superi 254. Le caratteristiche definite per l'archivio vengono scritte su un file di tipo sequenziale di nome STRUTTURA; questo file viene letto dal programma di gestione dell'archivio all'inizio dell'elaborazione e gli fornisce le costanti, che per ARCHIRANDOM sono incorporate nel programma stesso.
.preparare un programma, di nome INIZIOFILE, che, ricavando le informazioni necessarie dal file STRUTTURA, crea sul floppy un file sequenziale di tipo USR,

riservando tutti i settori che servono per il file random, e inoltre crea a priori il file sequenziale INDI1, lasciando vuote le chiavi, ma scrivendo gli indirizzi di traccia e settore utilizzati.
.preparare un programma, di nome ARANDOMUSER, che gestisce il file random, utilizzando il file STRUTTURA e il file INDI1.

```
1 REM STRUTTFILE
3 REM **********
5 S1$="LA SOMMA DELLE LUNGHEZZE SUPERA 254"
7 S2$="MODIFICA QUALCHE CAMPO"
9 PRINT"{CLR}DEFINIZIONE STRUTTURA RECORD{DOWN}"
11 REM CHIEDE NUMERO CAMPI MINORE DI 20
13 REM ************************************
15 INPUT"NOME FILE: ";N$
17 INPUT"QUANTI CAMPI: ";NC
19 IFNC<2ORNC>20THEN9
21 PRINT"{CLR}NOMI E LUNGHEZZE CAMPI"
23 PRINT"OGNI CAMPO MINORE DI 80 CARATT."
25 PRINT"OGNI NOME MASSIMO 10 CARATTERI"
27 DIMD$(NC),L(NC)
29 FORK=1TONC
31 PRINTK;"NOME: ";:INPUTD$(K)
33 INPUT"LUNGHEZZA: ";L(K)
35 NEXTK
37 PRINT"{CLR}CONTROLLO CAMPI"
39 FORK=1TONC
41 IFLEN(D$(K))>10THEND$(K)=LEFT$(D$(K),10)
43 IFL(K)>79THENL(K)=79
45 PRINTK;D$(K);" ";L(K)
47 NEXTK
49 PRINT"{DOWN}CONFERMI S/N ";:INPUTR$
51 IFR$="S"THEN61
53 INPUT"QUALE CAMPO: ";X
55 IFX<0ORX>NCTHEN53
57 INPUT"CAMPO: ";D$(X)
59 INPUT"LUNGHEZZA: ";L(X):GOTO37
61 REM CALCOLA LUNGHEZZA RECORD
63 REM *************************
65 S=NC:FORK=1TONC
67 S=S+L(K):NEXTK
69 IFS>254THENPRINTS1$:PRINTS2$:GOTO39
71 PRINT"NOME FILE: ";N$
73 PRINT"LUNGHEZZA RECORD: ";S
75 CLOSE15:OPEN15,8,15,"I"
76 PRINT#15,"S:STRUTTURA"
77 OPEN2,8,2,"STRUTTURA,S,W"
79 PRINT#2,N$
81 PRINT#2,NC
83 FORK=1TONC:PRINT#2,D$(K):NEXTK
85 FORK=1TONC:PRINT#2,L(K):NEXTK
87 CLOSE2
89 PRINT"{CLR}VERIFICA"
91 OPEN2,8,2,"STRUTTURA,S,R"
93 INPUT#2,A$
95 PRINT"NOME FILE: ";A$
97 INPUT#2,X
99 PRINT"NUMERO CAMPI: "X
```

```
101 FORK=1TOX:INPUT#2,D$(K):NEXTK
103 FORK=1TOX:INPUT#2,L(K):NEXTK
105 CLOSE2:CLOSE15
107 FORK=1TOX:PRINTK;D$(K);L(K):NEXTK
109 STOP
```

COMMENTO A STRUTTFILE

.15: chiede il nome del file random e lo pone in N$.

.17: chiede il numero NC dei campi e controlla che sia compreso tra 3 e 20. Abbiamo imposto il minimo 2, dato che per il nostro file la chiave di ordinamento lavora sui primi 2 campi. Il limite superiore 20 ci consente di vedere in un solo quadro video i campi.

.21/35: chiede il nome e la lunghezza dei campi, con la limitazione che il nome non superi 10 caratteri e la lunghezza 79 caratteri (sempre per la compatibilita' con altri modelli).

.37/47: controlla i dati e li aggiusta ai limiti se necessario.

.49/59: chiede conferma dei dati e consente di correggerli.

.61/69: controlla la lunghezza del record, aggiungendo alla lunghezza di ogni campo 1 per il CHR$(13) di fine campo. Se non risulta <=254 chiede di modificare la struttura del record.

.71/87: scrive il file STRUTTURA sul dischetto del programma, dopo aver cancellato, se esiste, un precedente file.

.89/109: rilegge il file STRUTTURA per verificare i dati e li ripropone sul video.

```
1 REM INIZIOFILE
3 REM **********
5 REM DIMENSIONA COSTANTI E VARIABILI
7 REM LEGGE FILE STRUTTURA
9 REM ********************************
11 CH$=CHR$(13)
13 S1$=" ":FORK=1TO39:S1$=S1$+" ":NEXTK
15 S2$="      "
17 OPEN15,8,15,"I"
19 OPEN2,8,2,"STRUTTURA,S,R":GOSUB229
21 INPUT#2,N$,NC:GOSUB229
23 N$=LEFT$(N$+S1$,16)
25 PRINT"DATI FILE STRUTTURA"
27 PRINT"FILE: ";N$;" NUMERO CAMPI: ";NC
29 INPUT"QUANTI RECORD: ";N
31 IFN>100THEN STOP
33 NX$=STR$(N)
35 NX$=RIGHT$(S2$+NX$,6)
37 DIMD$(NC),L(NC),Y$(NC)
39 FORK=1TONC:INPUT#2,D$(K):NEXTK
```

```
41 FORK=1TONC:INPUT#2,L(K):NEXTK
43 CLOSE2
45 FORK=1TONC:PRINTD$(K);"    ";L(K):NEXTK
47 SR=NC:FORK=1TONC:SR=SR+L(K):NEXTK
49 PRINT"LUNGHEZZA RECORD: ";SR
51 IFSR>254THENPRINT"NON POSSO PROSEGUIRE":STOP
53 REM CAMPI DATI CON CHR$(99) + SPAZI
55 REM ********************************
57 FORK=1TONC:Y$(K)=CHR$(99):IFL(K)=1THEN63
59 FORJ=1TOL(K)-1
61 Y$(K)=Y$(K)+CHR$(32):NEXTJ
63 NEXTK
65 IFSR=254THEN77
67 REM CREA CAMPO FINALE SE SR<254
69 REM ***************************
71 IFSR=253THENDN$="":GOTO77
73 DN$=CHR$(99):FORK=1TO253-SR-1
75 DN$=DN$+CHR$(32):NEXTK
77 REM CREA FILE USER
79 REM **************
81 PRINT"MONTA DISCO DATI"
83 PRINT"PREMI UN TASTO PER PROSEGUIRE"
85 R$="":GETR$:IFR$=""THEN85
87 PRINT"PAZIENZA, ATTENDI, STO PREPARANDO"
89 PRINT"IL DISCO DATI"
91 CLOSE15:OPEN15,8,15,"N0:"+N$+",99"
93 REM RIEMPIE RECORD
95 REM CON CAMPI CHE INIZIANO CON CHR$(99)
97 REM ***********************************
99 OPEN2,8,2,N$+",U,W":GOSUB229
101 REM SCRIVE PRIMO RECORD PER DATI DISCO
103 REM OCCUPANDO UN INTERO SETTORE
105 REM ***************************
107 PRINT#2,N$CH$"GGMMAA"CH$NX$CH$"0"CH$" "CH$" "CH$;
109 GOSUB229
111 PRINT#2,S1$CH$S1$CH$S1$CH$S1$CH$S1$CH$S2$
113 GOSUB229
115 FORK=1TON
117 FORJ=1TONC
119 PRINT#2,Y$(J);CH$;
121 GOSUB229
123 NEXTJ
125 IFSR<254THENPRINT#2,DN$
127 NEXTK
129 CLOSE2
131 REM CREA FILE INDICE
133 REM INDIRIZZO PRIMO SETTORE DIRECTORY
135 REM *********************************
137 CLOSE15:OPEN15,8,15,"I"
139 OPEN2,8,2,"#":GOSUB229
141 REM LEGGE SETTORE 18,1
143 REM *****************
145 PRINT#15,"U1:2,0,18,1":GOSUB229
147 REM PUNTATORE A TRACCIA E SETTORE
149 REM *****************************
151 PRINT#15,"B-P:2,3":GOSUB229
153 T$="":S$="":GET#2,T$,S$
155 T$=T$+CHR$(0):S$=S$+CHR$(0)
157 CLOSE2
159 T=ASC(T$):S=ASC(S$)
161 PRINT"PRIMO BLOCCO"
163 PRINT"TRACCIA: ";T;"  SETTORE: ";S
165 PRINT"PREMI UN TASTO PER PROSEGUIRE"
167 R$="":GETR$:IFR$=""THEN167
```

```
169 REM GENERA INDICE
171 REM LEGGENDO FILE USR COME RANDOM
173 REM ******************************
175 CLOSE15:OPEN15,8,15,"I"
177 OPEN2,8,2,"#":GOSUB229
179 OPEN3,8,3,"INDI1,S,W":GOSUB229
181 REM SCRIVE PRIMO RECORD INDI1
183 REM ************************
185 PRINT#3,Y$(1)+Y$(2);CH$;T;CH$;S:GOSUB229
187 REM SCRIVE ALTRI N RECORD
189 REM *******************
191 FORK=1TON
193 PRINT#15,"U1:2,0";T;S:GOSUB229
195 PRINT#15,"B-P:2,0":GOSUB229
197 GET#2,T$,S$:GOSUB229
199 T=ASC(T$+CHR$(0)):S=ASC(S$+CHR$(0))
201 PRINTT,S
203 PRINT#3,Y$(1)+Y$(2);CH$;T;CH$;S:GOSUB229
205 NEXTK:CLOSE2:CLOSE3
207 REM FILE STRUTTURA SU DISCO DATI
209 REM ****************************
211 OPEN2,8,2,"STRUTTURA,S,W":GOSUB229
213 PRINT#2,N$:GOSUB229
215 PRINT#2,NC:GOSUB229
217 FORK=1TONC:PRINT#2,D$(K):GOSUB229:NEXTK
219 FORK=1TONC:PRINT#2,L(K):GOSUB229:NEXTK
221 CLOSE2
223 PRINT"IL DISCO DATI E' PRONTO"
225 CLOSE15
227 OPEN15,8,15,"V":CLOSE15:STOP
229 REM ROUTINE ERRORE
231 REM *************
233 INPUT#15,EN,EM$,ET,ES
235 IFEN=0THENRETURN
237 PRINT"ERRORE DISCO":PRINTEN,EM$,ET,ES
239 STOP
```

COMMENTO A INIZIOFILE

.1/51: definisce costanti e variabili, legge il file STRUTTURA dal disco dei programmi, chiede il numero N dei record e controlla che non superi 100 (linea 31, potrai eventualmente modificare tale numero). Controlla che la somma delle lunghezze dei campi non superi 254 e se va bene prosegue.

.53/75: prepara i campi dummy con primo carattere CHR$(99).

.77/91: chiede di montare il disco dati e lo formatta.

.93/129: apre il file sequenziale di tipo USR, scrive i dati generali del disco sul primo settore e poi tutti i settori necessari per gli N record, chiude il file.

.131/167: per poter creare il file INDI1 va a leggere dalla directory l'indirizzo del primo blocco del file USR generato.

.169/227: crea il file INDI1, con chiavi dummy e gli indirizzi di traccia e settore, che ricava dai primi due byte di concatenamento dei settori del file sequenziale creato. Scrive sul disco dei dati il file STRUTTURA. Alla fine esegue il comando VALIDATE per essere sicuro della corrispondenza della BAM alla situazione creata. Il primo record di INDI1 contiene l'indirizzo del settore del disco usato per le informazioni generali sull'archivio.
.229/239: routine errore disco.
Questo programma, a parte la sua utilita' per realizzare il nostro archivio, e' molto interessante per imparare a maneggiare bene le registrazioni che si trovano sul floppy.
Se chiedi la lista della directory del floppy, vedrai registrati 3 file, quello di tipo USR con il nome che gli hai assegnato, INDI1 e STRUTTURA.

```
1 REM ARANDOMUSER
3 REM ***********
5 REM DEFINIZIONE COSTANTI E VARIABILI
7 REM ********************************
9 S1$="DATA ULTIMO AGG.        "
11 S2$="FINITO SPAZIO ASSEGNATO"
13 SP$="":FORK=1TO79:SP$=SP$+" ":NEXTK
15 CH$=CHR$(13):LIM$=CHR$(99)+CHR$(32)+CHR$(32)
17 REM LEGGE DATI DA FILE STRUTTURA
19 REM ****************************
21 PRINT"[CLR][DOWN][DOWN]"TAB(8)"MONTA DISCO DATI"
23 GOSUB257
25 OPEN15,8,15,"I"
27 OPEN2,8,2,"STRUTTURA,S,R":GOSUB219
29 INPUT#2,N$,NC:GOSUB219
31 DIMD$(NC):DIMY$(NC):DIML(NC)
33 FORK=1TONC:INPUT#2,D$(K):NEXTK
35 FORK=1TONC:INPUT#2,L(K):NEXTK
37 CLOSE2
45 REM LEGGE DATI DISCO
47 REM ****************
49 OPEN10,8,10,"INDI1,S,R":GOSUB219
51 REM SETTORE DATI GENERALI
53 REM *********************
55 INPUT#10,A$,TD,SD
57 GOSUB187
59 REM DIMENSIONAMENTI PER INDICE
61 REM **************************
63 DIMC$(N),T%(N),S%(N)
65 REM LETTURA INDICE
67 REM **************
69 GOSUB213:GOSUB231
71 REM PRESENTAZIONE MENU' E SCELTE
73 REM ****************************
75 PRINT"[CLR][DOWN][DOWN]";TAB(10);"GESTIONE ARCHIVIO":PRINT
77 PRINTTAB(10)"1=AGGIORNAMENTO"
79 PRINTTAB(10)"2=LISTA PRINCIPALE"
81 PRINTTAB(10)"3=CREAZ.IND.SEC."
```

```
83 PRINTTAB(10)"4=LISTA SECONDARIA"
85 PRINTTAB(10)"9=FINE"
87 INPUT"COSA SCEGLI ";X
89 IFX<1ORX>4ANDX<>9THENPRINT"]";:GOTO87
91 PRINT:PRINT"PRESENTI ";K;" RECORD"
93 PRINTG1$;"/";M1$;"/";A1$
95 GOSUB257
97 IFX=9THEN497
99 ONXGOTO295,427,459,479
101 STOP
103 REM INGRESSO DATI
105 REM **************
107 PRINT"]INGRESSO DATI"
109 FORJ=1TONC:Y$(J)="":NEXTJ
111 PRINT"PER USCIRE $ PER COGNOME";
113 PRINT"▮SE MANCANO DATIPREMI SOLO RETURN"
115 PRINTD$(1);:INPUTY$(1)
117 IFY$(1)="$"THENW=1:RETURN
119 FORJ=2TONC:PRINTD$(J);:INPUTY$(J):NEXTJ
121 REM SISTEMA LUNGHEZZA DATI
123 REM ************************
125 FORJ=1TONC:Y$(J)=LEFT$(Y$(J)+SP$,L(J))
127 NEXTJ:GOSUB371:RETURN
129 REM SCRITTURA INDICE PRINCIPALE
131 REM ******************************
133 PRINT#15,"S0:INDI1":PRINT#15,"I"
135 OPEN10,8,10,"INDI1,S,W":GOSUB219
137 PRINT#10,LIM$;CH$;TD;CH$;SD;CH$;:GOSUB219
139 FORJ=1TON
141 PRINT#10,C$(J);CH$;T%(J);CH$;S%(J);CH$;
143 GOSUB219:NEXTJ
145 CLOSE10:RETURN
147 REM SCRITTURA NEL BUFFER
149 REM **********************
151 FORJ=1TONC:Y$(J)=LEFT$(Y$(J)+SP$,L(J))
153 PRINT#11,Y$(J);CH$;:GOSUB219
155 NEXTJ:RETURN
157 REM LETTURA SETTORE NEL BUFFER
159 REM ******************************
161 PRINT#15,"U1:11,0";T;S:GOSUB219:RETURN
163 REM PUNTATORE NEL BUFFER
165 REM **********************
167 PRINT#15,"B-P:"11;2:GOSUB219:RETURN
169 REM SCRITTURA RECORD
171 REM ******************
173 PRINT#15,"U2:"11;0;T;S:GOSUB219:RETURN
175 REM LETTURA RECORD
177 REM ***************
179 PRINT#15,"I":OPEN11,8,11,"#":GOSUB219
181 PRINT#15,"U1:"11;0;T;S:GOSUB219:GOSUB163
183 FORJ=1TONC:INPUT#11,Y$(J):GOSUB219
185 NEXTJ:CLOSE11:RETURN
187 REM LETT. DATI DISCO
189 REM ******************
191 OPEN11,8,11,"#":GOSUB219
193 PRINT#15,"U1:"11;0;TD;SD:GOSUB219
195 GOSUB163:INPUT#11,N$,R$,N,K:GOSUB219
197 G1$=LEFT$(R$,2):M1$=MID$(R$,3,2)
199 A1$=RIGHT$(R$,2):CLOSE11:RETURN
201 REM LETT. INDICE
203 REM *************
205 CLOSE10:OPEN10,8,10,"INDI1,S,R":GOSUB219
207 INPUT#10,A$,TD,SD
209 REM DA SECONDO RECORD
```

```
211 REM *****************
213 FORJ=1TON
215 INPUT#10,C$(J),T%(J),S%(J):GOSUB219
217 NEXTJ:CLOSE10:RETURN
219 REM ROUTINE ERRORE
221 REM **************
223 INPUT#15,EN,EM$,ET,ES
225 IFEN=0THENRETURN
227 PRINT"■ERRORE DISCO"
229 PRINTEN,EM$,ET,ES:CLOSE15:STOP
231 REM DATA DISCO
233 REM **********
235 PRINT"DATA PER DISCO"
237 INPUT"  GG,MM,AA■■■■■■■■■■■■";G$,M$,A$:RETURN
239 REM SCRITT. IND. SEC.
241 REM *****************
243 PRINT#15,"S:INDI2":PRINT#15,"I"
245 OPEN10,8,10,"INDI2,S,W"
247 FORJ=1TOK
249 PRINT#10,C$(J);CH$;T%(J);CH$;S%(J);CH$;
251 GOSUB219:NEXTJ:CLOSE10:RETURN
253 REM ATTESA TASTO
255 REM ************
257 R$="":GETR$:IFR$=""THEN257
259 RETURN
261 REM RICERCA POSTO LIBERO
263 REM ********************
265 FORI=1TON
267 IFLEFT$(C$(I),1)<>CHR$(99)THEN271
269 T=T%(I):S=S%(I):JJ=I:I=N
271 NEXTI:RETURN
273 REM AGGIUNTA NUOVI RECORD
275 REM *********************
277 GOSUB103:GOSUB261
279 IFW=1THENK=K-1:CLOSE11:GOTO497
281 GOSUB157:GOSUB163:GOSUB147
283 REM AGGIORNAMENTO INDICE
285 REM ********************
287 C$(JJ)=Y$(1)+Y$(2)
289 GOSUB169:K=K+1
291 IFK>NTHENK=K-1:PRINTS2$:CLOSE11:GOTO497
293 GOTO277
295 REM AGGIORNAMENTO
297 REM *************
299 PRINTTAB(10);"■■■AGGIORNAMENTO ARCHIVIO"
301 PRINT:PRINTTAB(10);"1=CORREZIONE"
303 PRINTTAB(10);"2=AGGIUNTA ELEM."
305 PRINTTAB(10);"3=CANCELL.ELEM."
307 PRINTTAB(10);"9=FINE"
309 INPUT"COSA SCEGLI ";X:IFX=9THEN497
311 IFX<1ORX>3THEN299
313 IFX=2THEN393
315 IFX=3THEN409
317 GOTO351
319 REM RICERCA RECORD
321 REM **************
323 PRINT"■";D$(1);:Y$(1)="":INPUTY$(1):W=0
325 IFY$(1)="$"THENW=1:RETURN
327 PRINTD$(2);:Y$(2)="":INPUTY$(2)
329 Y$(1)=LEFT$(Y$(1)+SP$,L(1))
331 Y$(2)=LEFT$(Y$(2)+SP$,L(2))
333 INPUT"QUALE OCCORRENZA ";X
335 IFX<=0THENPRINT"■";:GOTO333
337 FORJ=1TON:IFC$(J)=Y$(1)+Y$(2)THEN345
```

```
339 NEXTJ
341 J=J-1:PRINT"NON TROVATO ";Y$(1);"  ";Y$(2)
343 GOSUB257:GOTO319
345 IFX<>1THENX=X-1:GOTO339
347 T=T%(J):S=S%(J)
349 I=J:J=N:NEXTJ:RETURN
351 REM CORREZIONE
353 REM **********
355 GOSUB319:IFW=1THEN299
357 GOSUB175
359 REM VA A MODIFICA DATI
361 REM ******************
363 GOSUB371
365 PRINT#15,"I":OPEN11,8,11,"#":GOSUB163:GOSUB147
367 C$(I)=Y$(1)+Y$(2):T%(I)=T:S%(I)=S
369 GOSUB169:CLOSE11:GOTO351
371 REM CONTROLLO DATI E MODIFICHE
373 REM **************************
375 PRINT"□CONTROLLO DATI E MODIFICHE"
377 FORJ=1TONC:PRINTJ;" ";D$(J);" ";Y$(J):NEXTJ
379 INPUT"■TUTTO BENE S/N ";X$:IFX$="S"THENRETURN
381 PRINT"QUALE CAMPO (0 USCITA)";:INPUTX
383 IFX=0THENRETURN
385 IFX>NCORX<1THENPRINT"⊐⊐":GOTO379
387 Y$(X)=""
389 INPUTY$(X):Y$(X)=LEFT$(Y$(X)+SP$,L(X))
391 GOTO375
393 REM AGGIUNTA
395 REM ********
397 OPEN11,8,11,"#":GOSUB219
399 PRINT"□AGGIUNTA NUOVI RECORD"
401 PRINT"PRESENTI ";K;" RECORD"
403 PRINT"PUOI AGGIUNGERE ";N-K;" RECORD"
405 GOSUB257
407 W=0:K=K+1:GOTO277
409 REM CANCELLAZIONE
411 REM *************
413 M=0
415 GOSUB319:IFW=1THEN423
417 X=L(1)+L(2):C$(I)=LEFT$(LIM$+SP$+SP$,X)
419 M=M+1
421 GOTO415
423 REM SIST. INDICE
425 GOSUB523:K=K-M:GOTO507
427 REM LISTA FILE
429 REM **********
431 T$="LISTA PER INDICE PRIM."
433 PRINT"ACCENDI STAMPANTE     PREMI UN TASTO"
435 GOSUB257
437 PRINT"□";T$:OPEN4,4
439 PRINT#4:PRINT#4:PRINT#4,T$
441 PRINT#4:PRINT#4
443 FOR1=1TOK:T=T%(1):S=S%(1):GOSUB175
445 PRINT#4,Y$(1);"  ";Y$(2)
447 IFNC=2THEN451
449 FORM=3TONC:PRINT#4,D$(M)":  "Y$(M):NEXTM
451 PRINT#4:PRINT#4
455 NEXTI
457 CLOSE4:GOTO517
459 REM CREAZIONE INDICE SECONDARIO
461 REM ***************************
463 PRINT"□INDICE SECONDARIO"
465 INPUT"■QUALE CAMPO ";X$
467 X=VAL(X$):IFX<2ORX>NCTHENGOTO465
```

```
469 FORI=1TOK:T=T%(I):S=S%(I):GOSUB175
470 IFLEN(Y$(X))=0THENY$(X)=CHR$(160)
471 C$(I)=Y$(X):NEXTI:GOSUB551
473 GOSUB239
475 PRINT"FINITO IND. SEC."
477 GOTO517
479 REM LISTA PER INDICE SECONDARIO
481 REM ***************************
483 PRINT"]LISTA IND. SEC."
485 OPEN10,8,10,"INDI2,S,R":GOSUB219
487 FORJ=1TOK:INPUT#10,C$(J),T%(J),S%(J)
489 GOSUB219:NEXTJ
491 CLOSE10
493 T$="LISTA IND. SEC. "
495 GOTO433
497 REM CHIUSURA
499 REM ********
501 PRINT"CHIUSURA ARCHIVIO"
503 PRINT"IND. PRINC. DA ORDIN. S/N ":INPUTR$
505 IFR$="S"THENGOSUB523
507 GOSUB129:OPEN11,8,11,"#":GOSUB219
509 PRINT#15,"U1:"11;0;TD;SD:GOSUB219:GOSUB163
511 PRINT#11,N$CH$G$M$A$CH$;N;CH$;K;CH$;
513 GOSUB219
515 PRINT#15,"U2:"11;0;TD;SD:GOSUB219:CLOSE11
517 PRINT"FINITO AGGIORNAMENTO"
519 PRINT"SONO PRESENTI ";K;" RECORD"
521 CLOSE15:STOP
523 REM ORDINAMENTO CRESCENTE
525 REM *********************
527 L=N-1
529 W=0
531 FORJ=1TOL
533 IFC$(J)<=C$(J+1)THEN543
535 R$=C$(J):C$(J)=C$(J+1):C$(J+1)=R$
537 X=T%(J):T%(J)=T%(J+1):T%(J+1)=X
539 X=S%(J):S%(J)=S%(J+1):S%(J+1)=X
541 W=1
543 NEXTJ
545 IFW=0THENRETURN
547 IFL=1THENRETURN
549 L=L-1:GOTO529
551 REM ORDINAMENTO DECRESCENTE
553 REM ***********************
555 L=K-1
557 W=0
559 FORJ=1TOL
561 IFC$(J)>=C$(J+1)THEN571
563 R$=C$(J):C$(J)=C$(J+1):C$(J+1)=R$
565 X=T%(J):T%(J)=T%(J+1):T%(J+1)=X
567 X=S%(J):S%(J)=S%(J+1):S%(J+1)=X
569 W=1
571 NEXTJ
573 IFW=0THENRETURN
575 IFL=1THENRETURN
577 L=L-1:GOTO557
```

Non ci sembra necessario riportare il commento di ARANDOMUSER. Esso e' infatti stato ricavato da ARCHIRANDOM, apportando alcune modifiche che elenchiamo:

. il settore che contiene i dati generali del disco non e' piu' quello di indirizzo 1,0; il suo indirizzo si trova nel primo record di INDI1.

. i dati generali del disco non comprendono piu' l'indirizzo dell'ultimo settore scritto, ma e' presente il numero N dei record previsti e il numero K di quelli effettivamente presenti.

. il file INDI1 deve essere trasferito nei 3 vettori ad esso riservati a partire dal secondo record.

. il file INDI1 deve essere sempre ordinato e riscritto tutto, altrimenti si perdono gli indirizzi di traccia e settore.

. i record di INDI1 non usati, chiave che inizia con CHR$(99), nell'ordinamento vanno in fondo, ma restano.

. per cancellare un record si pone CHR$(99) all'inizio della chiave in INDI1.

. se si crea INDI2 si deve lavorare su INDI1 ordinato, INDI2 viene preparato solo per i K record presenti.

. nella fase di aggiunta record si deve cercare in INDI1 la prima chiave che inizia con CHR$(99), leggere il corrispondente settore e riscriverlo senza perdere i primi due caratteri di concatenamento.

. nel programma non e' piu' presente la fase di inizializzazione, svolta dal programma INIZIOFILE.

Se hai riflettuto bene su questo argomento, non ti sara' difficile adattare questi programmi per poter gestire anche record logici di lunghezza sottomultipla di un settore; solo che in questo caso l'indice deve contenere anche un campo che fornisca la posizione del record logico nel settore, da caricare nel puntatore al buffer.

3.7 FILE RELATIVI DI DATI

I file di questo tipo sono trattati dalle istruzioni del DOS per la gestione dei file relativi, elencate nel Paragrafo 3.2.

La differenza tra i file random e i file relativi consiste nei seguenti fatti:

. per accedere ai record di un file relativo si deve

fornire al programma il numero d'ordine del record nel file, indirizzo logico, invece dell'indirizzo fisico del blocco.

. i record logici devono essere di lunghezza fissa, che deve essere al massimo di 254 caratteri, comprendendo i caratteri di fine campo e/o record. Pero' la lunghezza puo' essere anche un numero che non risulti sottomultiplo di 254, infatti il sistema gestisce anche record logici registrati a cavallo tra due settori (spanned), come per i file sequenziali.
I primi due caratteri di ogni settore sono usati, come al solito, per concatenare tra loro i settori.

. il sistema, conoscendo la lunghezza di un record e il suo numero d'ordine nel file, ricava con un semplice algoritmo l'indirizzo di traccia e settore necessario.

. il sistema gestisce i file relativi con l'aiuto di un suo indice interno, che genera al momento della creazione del file, chiamato SIDE SECTOR. Esso e' formato al massimo da 6 settori numerati logicamente da 0 a 5, che vengono utilizzati per contenere l'indice di tutti i blocchi fisici utilizzati per il file relativo. In ogni blocco dell'indice SIDE SECTOR possono essere registrati gli indirizzi di 120 settori; per questa ragione il file relativo puo' occupare al massimo 120*6=720 settori, che nel nostro caso sono di piu' di quelli disponibili su un floppy. Inoltre il numero dei record logici gestibili non puo' superare 65535, infatti e' espresso in due byte consecutivi. La struttura di un blocco SIDE SECTOR e' la seguente:

. byte 0 e 1, concatenamento al settore successivo.
. byte 2, numero del blocco, da 0 a 5.
. byte 3, lunghezza del record logico, massimo 254.
. byte 4 e 5, traccia e settore del SIDE SECTOR 0.
. byte 6 e 7, traccia e settore del SIDE SECTOR 1.
. byte 8 e 9, traccia e settore del SIDE SECTOR 2.
. byte 10 e 11, traccia e settore del SIDE SECTOR 3.
. byte 12 e 13, traccia e settore del SIDE SECTOR 4.
. byte 14 e 15, traccia e settore del SIDE SECTOR 5.
. byte 16 e 17, indirizzo primo settore dati (t,s).
. byte 18 e 19, indirizzo secondo settore dati (t,s).
.....
.....

. byte 254 e 255, indirizzo 120-esimo settore dati (t,s).
. viene registrata una entrata nella directory che tiene conto di tutti i blocchi occupati, file e side sector. Inoltre nell'entrata e' indicato l'indirizzo del primo settore del side sector.
. il file puo' essere creato scrivendo i record in ordine casuale; in conseguenza possono restare dei record non usati, che il sistema riempie scrivendo 255 (FFH) nella prima posizione e tutti bit 0 nelle altre.
. il sistema usa un buffer in piu' per gestire il file, infatti deve leggere anche il side sector.
. e' consigliabile preestendere il file in fase di creazione; questo rende piu' veloci le operazioni successive. Per preestendere il file basta scrivere l'ultimo record; infatti pensa il sistema a predisporre tutti i precedenti e a creare il numero di blocchi di side sector necessari.

Anche per i file relativi sussiste il problema dell'accesso ai record, gia' esaminato per i file random. L'indice SIDE SECTOR creato dal sistema serve solo per gestire i blocchi fisici. In conseguenza un buon programma di gestione di un archivio deve creare anche un indice sequenziale per chiavi di accesso; in questo caso l'indice puo' essere formato solo da due campi: chiave di accesso, numero d'ordine del record nel file.

Seguono 4 programmi esempio che consentono di capire come si usano i comandi del DOS per i file relativi; essi sono:
. FRELCREA, per preestendere un file,
. FRELSCRIVI, per scrivere record in un file,
. FRELLEGGI, per leggere record da un file,
. FRELCAMPOLEGGI, per mostrare come si puo' accedere a un campo e non a tutto il record.
Questi programmi gestiscono un file relativo di nome PROVA REL, formato da 30 record logici di 21 caratteri ciascuno. Ogni record comprende 2 campi, il primo di 10 caratteri + fine campo (CHR$(13)), e il secondo di 9 caratteri + fine campo.
Dopo aver fatto girare i singoli programmi puoi analizzare il contenuto della directory e dei blocchi occupati per verificare quanto sopra esposto, usando i programmi di utilita' DCOMEFS e TRAC/SET.
Osserva in ogni programma come avviene la preparazione del numero del record e del puntatore al campo.

```
1 REM FRELCREA
3 REM ********
5 REM CREAZIONE FILE RELATIVO
7 REM PREESTENSIONE FILE
9 REM 30 RECORD DI 21 CARATTERI
11 REM COMPRESO RETURN FINALE
13 REM COMPRESO RETURN FINALE
15 REM *************************
17 DR$="0":NF$=DR$+":PROVA REL,L,"
19 SA=2:LF=2:RC$="----------":RC$=RC$+RC$
21 BI=1:NR=30:LU=21
23 OPEN15,8,15,"I"
25 REM APERTURA FILE
27 REM *************
29 OPENLF,8,SA,NF$+CHR$(LU)
31 GOSUB73
33 REM PREESTENSIONE FILE
35 REM PER NR RECORD
37 REM ******************
39 FORX=NRTO1STEP-1
41 HI=INT(X/256):LO=X-HI*256
43 C$="P"+CHR$(SA)
45 REM NUMERO RECORD (BYTE BASSO + BYTE ALTO)
47 REM **************************************
49 C$=C$+CHR$(LO)+CHR$(HI)
51 REM PUNTATORE AL PRIMO BYTE DEL RECORD
53 REM **********************************
55 C$=C$+CHR$(BI)
57 REM POSIZIONA IL PUNTATORE
59 REM **********************
61 PRINT#15,C$:GOSUB73
63 REM SCRIVE IL RECORD DI LINEETTE
65 REM ****************************
67 PRINT#LF,RC$:GOSUB73
69 NEXTX:CLOSELF:GOSUB73:CLOSE15:END
71 REM ROUTINE ERRORE
73 REM **************
75 INPUT#15,EN,EM$,ET,ES
77 IFEN=0OREN=50THENRETURN
79 PRINTEN;EM$;ET;ES
81 STOP:RETURN
```

```
1 REM FRELSCRIVI
3 REM **********
5 REM SCRITTURA RECORD NEL FILE
7 REM RELATIVO, OGNI RECORD 2 CAMPI
9 REM NOME DI 10 CARATTERI
11 REM TELEFONO DI 9 CARATTERI
13 REM LUNGHEZZA RECORD=10+1+9+1=21
15 REM ****************************
17 DR$="0":NF$=DR$+":PROVA REL,L,"
19 LU=21:SA=2:LF=2:B$="          "
21 BI=1:NR=30
23 OPEN15,8,15,"I"
25 REM APERTURA FILE
27 REM *************
29 OPENLF,8,SA,NF$+CHR$(LU)
```

```
31 GOSUB71
33 PRINT"NOME=* PER USCIRE":PRINT
35 INPUT"NOME(10)";N$
37 IFN$="*"THEN69
39 N$=LEFT$(N$+B$,10)
41 INPUT"TEL. (9)";T$:T$=LEFT$(T$+B$,9)
43 INPUT"RECORD ";R
45 IFR=0ORR>NRTHENPRINTCHR$(145);:GOTO43
47 REM PREPARAZIONE PUNTATORE
49 REM **********************
51 HI=INT(R/256):LO=R-HI*256
53 C$="P"+CHR$(SA)
55 C$=C$+CHR$(LO)+CHR$(HI)
57 C$=C$+CHR$(BI)
59 PRINT#15,C$:GOSUB71
61 REM SCRIVE RECORD
63 REM *************
65 PRINT#LF,N$;CHR$(13);T$:GOSUB71
67 GOTO33
69 CLOSELF:GOSUB71:CLOSE15:END
71 REM ROUTINE ERRORE, ACCETTA COD. 50
73 REM ********************************
75 INPUT#15,EN,EM$,ET,ES
77 IFEN=0OREN=50THENRETURN
79 PRINTEN;EM$;ET;ES
81 STOP:RETURN
```

```
1 REM FRELLEGGI
3 REM *********
5 REM LETTURA RECORD
7 REM **************
9 DR$="0":NF$=DR$+":PROVA REL,L,"
11 LU=21:SA=2:LF=2
13 BI=1:NR=30
15 OPEN15,8,15,"I"
17 REM APRE FILE
19 REM *********
21 OPENLF,8,SA,NF$+CHR$(LU)
23 GOSUB61
25 PRINT"RECORD=0 PER USCIRE":PRINT
27 INPUT"RECORD ";R
29 IFR=0THEN59
31 IFR<0ORR>NRTHENPRINTCHR$(145);:GOTO27
33 REM PREPARA PUNTATORE
35 REM *****************
37 HI=INT(R/256):LO=R-HI*256
39 C$="P"+CHR$(SA)
41 C$=C$+CHR$(LO)+CHR$(HI)
43 C$=C$+CHR$(BI)
45 PRINT#15,C$:GOSUB61
47 REM LEGGE RECORD
49 REM ************
51 INPUT#LF,N$,T$:GOSUB61
53 PRINT"NOME: ";N$
55 PRINT"TEL.: ";T$
57 GOTO25
59 CLOSELF:GOSUB61:CLOSE15:END
```

```
61 REM ROUTINE ERRORE, AMMETTE COD. 50
63 REM ********************************
65 INPUT#15,EN,EM$,ET,ES
67 IFEN=0OREN=50THENRETURN
69 PRINTEN;EM$;ET;ES
71 STOP:RETURN
```

```
1 REM FRELCAMPOLEGGI
3 REM **************
5 REM LETTURA SOLO DEL SECONDO CAMPO
7 REM DI UN RECORD R
9 REM ******************************
11 DR$="0":NF$=DR$+":PROVA REL,L,"
13 LU=21:SA=2:LF=2
15 BI=12:NR=30
17 INPUT"RECORD ";R
19 IFR=0ORR>NRTHENPRINTCHR$(145);:GOTO15
21 OPEN15,8,15,"I"
23 REM APREFILE
25 REM ********
27 OPENLF,8,SA,NF$+CHR$(LU)
29 GOSUB55
31 REM PREPARA PUNTATORE
33 REM *****************
35 HI=INT(R/256):LO=R-HI*256
37 C$="P"+CHR$(SA)
39 C$=C$+CHR$(LO)+CHR$(HI)
41 C$=C$+CHR$(BI)
43 PRINT#15,C$:GOSUB55
45 REM LEGGE CAMPO
47 REM ***********
49 INPUT#LF,T$:GOSUB55
51 PRINT"TEL.:  ";T$
53 CLOSELF:GOSUB55:CLOSE15:END
55 REM ROUTINE ERRORE, AMMETTE COD. 50
57 REM ********************************
59 INPUT#15,EN,EM$,ET,ES
61 IFEN=0OREN=50THENRETURN
63 PRINTEN;EM$;ET;ES
65 STOP:RETURN
```

Ti facciamo notare che la routine di errore scarta il codice 50, che in questo caso non e' un errore.
Il programma FLCREA preestende il file senza dividere il record in campi. Il puntatore deve avere il valore 1 per iniziare a scrivere il record.
Nel programma FRELSCRIVI il record viene scritto con una sola PRINT# che ha nella lista-dati tutti i campi con i loro separatori.
Il programma FRELLEGGI legge i due campi del record con una sola INPUT# in due variabili.
Nel programma FRELCAMPOLEGGI invece il puntatore viene posto a 12 per leggere solo il secondo campo del record.

Abbiamo modificato il programma ARCHIRANDOM per ottenere ARCHIRELATIVO, che gestisce un file relativo con indice principale e secondario. In questo caso i dati generali dell'archivio, che erano memorizzati nel settore 0 della traccia 1, sono memorizzati in un piccolo file sequenziale di nome DATIGEN, sul disco dati, che viene creato quando si crea ex-novo l'archivio, e, ad ogni elaborazione, viene letto all'inizio e aggiornato alla fine. Per il resto abbiamo lasciato invariata la struttura del record e dei campi. Inoltre, nella fase di creazione dell'archivio, i record sono caricati in sequenza, senza chiedere il numero del record, come e' stato fatto in FRELSCRIVI.

```
1 REM ARCHIRELATIVO
3 REM **************
5 REM DEFINIZIONE COSTANTI E VARIABILI
7 REM ***********************************
9 NC=14:REM NUMERO CAMPI RECORD
11 S1$="DATA ULTIMO AGG.      "
13 S2$="FINITO SPAZIO ASSEGNATO"
15 SP$="":FORK=1TO79:SP$=SP$+" ":NEXTK
17 SWW=0:REM PER EVITARE RIDIMENSIONAMENTO
19 CH$=CHR$(13):LIM$=CHR$(99)+CHR$(99)+CHR$(99)
21 DIMD$(NC):DIMY$(NC):DIML(NC)
23 DATA"COGNOME:","NOME    :","INDIR. :"
25 DATA"CITTA'  :","PROV.   :","TELEF. :"
27 DATA"L.NASC.:","D.NASC.:","T.STUD.:"
29 DATA"OCC.AT.:","OCC.PR.:","ST.CIV.:"
31 DATA"NOTA 1  :","NOTA 2  :"
33 DATA20,15,30,25,11,10,20,8,20,20,20,1,20,20
35 FORJ=1TONC:READD$(J):NEXTJ
37 FORJ=1TONC:READL(J):NEXTJ
39 REM PRESENTAZIONE MENU' E SCELTE
41 REM ******************************
```

```
43 CLOSE15:OPEN15,8,15
45 PRINT"☐██";TAB(10);"GESTIONE ARCHIVIO":PRINT
47 PRINTTAB(10)"1=INIZIO EX-NOVO"
49 PRINTTAB(10)"2=AGGIORNAMENTO"
51 PRINTTAB(10)"3=LISTA PRINCIPALE"
53 PRINTTAB(10)"4=CREAZ.IND.SEC."
55 PRINTTAB(10)"5=LISTA SECONDARIA"
57 PRINTTAB(10)"9=FINE"
59 INPUT"COSA SCEGLI ";X
61 IFX<1ORX>5ANDX<>9THENPRINT"☐";:GOTO59
63 PRINT:GOSUB297
65 R$="":PRINTTAB(10)"MONTA DISCO DATI"
67 GETR$:IFR$=""THEN67
69 IFX=1THEN91
71 PRINT#15,"I"
73 GOSUB259:PRINT"PRESENTI ";K;" RECORD"
75 NN$=N$+",L,"+CHR$(254)
77 PRINTS1$;G1$;"/";M1$;"/";A1$
79 R$="":GETR$:IFR$=""THEN79
81 IFSWW=0THENDIMC$(N),T%(N)
83 GOSUB273
85 IFX=9THEN561
87 ONX-1GOTO361,491,523,543
89 STOP
91 REM INIZIO EX-NOVO ARCHIVIO
93 REM ************************
95 PRINT"POSSO FORMATTARE IL DISCO DATI S/N"
97 R$="":INPUTR$:IFR$=""THEN97
99 IFR$="S"THEN103
101 STOP
103 GOSUB323
105 INPUT"█NOME FILE DA INIZIARE: ";N$
107 INPUT"QUANTI RECORD IN TUTTO: ";N
109 DIMC$(N),T%(N):SWW=1
111 REM PREPARAZIONE CAMPI DUMMY
113 REM **************************
115 P1$="*":FORJ=1TO52:P1$=P1$+" ":NEXTJ
117 P2$=LEFT$(P1$,37)
119 REM PREESTENSIONE FILE N$
121 REM ************************
123 OPEN11,8,11,N$+",L,"+CHR$(254)
125 FORJ=NTO1STEP-1
127 HI=INT(J/256):LO=J-HI*256
129 C$="P"+CHR$(11)+CHR$(LO)+CHR$(HI)+CHR$(1)
131 PRINT#15,C$:GOSUB285
133 PRINT#11,P1$CH$P1$CH$P1$CH$P1$CH$P2$CH$;
135 GOSUB285
137 NEXTJ:CLOSE11:GOSUB285
139 REM PREPARAZIONE INDICE
141 REM **********************
143 FORJ=1TON:T%(J)=J
145 C$(J)=LIM$:NEXTJ
147 GOSUB177:REM SCRIVE INDI1
149 K=0:GOSUB221:GOTO43
151 REM INGRESSO DATI
153 REM **************
155 PRINT"☐INGRESSO DATI"
157 FORJ=1TONC:Y$(J)="":NEXTJ
159 PRINT"PER USCIRE $ PER COGNOME";
161 PRINT"█SE MANCANO DATIPREMI SOLO RETURN"
163 PRINTD$(1);:INPUTY$(1)
165 IFY$(1)="$"THENW=1:RETURN
167 FORJ=2TONC:PRINTD$(J);:INPUTY$(J):NEXTJ
169 REM SISTEMA LUNGHEZZA DATI
```

```
171 REM ************************
173 FORJ=1TONC:Y$(J)=LEFT$(Y$(J)+SP$,L(J))
175 NEXTJ:GOSUB439:RETURN
177 REM SCRITTURA INDICE PRINCIPALE
179 REM *****************************
181 PRINT#15,"S0:INDI1":PRINT#15,"I"
183 OPEN10,8,10,"INDI1,S,W":GOSUB285
185 FORJ=1TON
187 PRINT#10,C$(J);CH$;T%(J);CH$;
189 GOSUB285:NEXTJ
191 CLOSE10:RETURN
193 REM SCRITTURA RECORD PUNTATO
195 REM COSTRUZIONE STRINGA DI STAMPA
197 REM ********************************
199 C$="":FORJ=1TONC
201 Y$(J)=LEFT$(Y$(J)+SP$,L(J))
203 C$=C$+Y$(J)+CH$:NEXTJ
205 PRINT#11,C$;:GOSUB285
207 RETURN
209 REM DEFINISCE POSIZIONE RECORD K
211 REM ******************************
213 HI=INT(T/256)
215 LO=T-HI*256
217 PRINT#15,"P"+CHR$(11)+CHR$(LO)+CHR$(HI)+CHR$(1
219 GOSUB285:RETURN
221 REM SCRITTURA DATIGEN DISCO
223 REM ************************
225 PRINT#15,"S0:DATIGEN"
227 PRINT#15,"I"
229 OPEN10,8,10,"DATIGEN,S,W":GOSUB285
231 PRINT#10,N$CH$G$M$A$CH$;N;CH$;K;CH$;
233 CLOSE10:RETURN
235 REM RICERCA POSTO NELL'INDICE
237 REM ***************************
239 JJ=0:FORJ=1TON
241 IFLEFT$(C$(J),1)=CHR$(99)THEN 245
243 NEXTJ:RETURN
245 JJ=J:J=N:GOTO243
247 REM LETTURA RECORD
249 REM **************
251 PRINT#15,"I":OPEN11,8,11,NN$:GOSUB285
253 GOSUB209
255 FORJ=1TONC:INPUT#11,Y$(J):GOSUB285
257 NEXTJ:CLOSE11:RETURN
259 REM LETT.DATI DISCO
261 REM ***************
263 OPEN10,8,10,"DATIGEN,S,R":GOSUB285
265 INPUT#10,N$,R$,N,K:GOSUB285
267 CLOSE10:GOSUB285
269 G1$=LEFT$(R$,2):M1$=MID$(R$,3,2)
271 A1$=RIGHT$(R$,2):RETURN
273 REM LETT. INDICE
275 REM ************
277 OPEN10,8,10,"INDI1,S,R":GOSUB285
279 FORJ=1TON
281 INPUT#10,C$(J),T%(J):GOSUB285
283 NEXTJ:CLOSE10:RETURN
285 REM ROUTINE ERRORE
287 REM **************
289 INPUT#15,EN,EM$,ET,ES
291 IFEN=0OREN=50THENRETURN
293 PRINT"■ERRORE DISCO"
295 PRINTEN,EM$,ET,ES:CLOSE15:STOP
297 REM DATA DISCO
```

```
299 REM **********
301 PRINT"DATA PER DISCO"
303 INPUT"  GG,MM,AA████████████";G$,M$,A$:RETURN
305 REM SCRITT. IND. SEC.
307 REM *****************
309 PRINT#15,"S:INDI2":PRINT#15,"I"
311 OPEN10,8,10,"INDI2,S,W"
313 FORJ=1TOK
315 PRINT#10,C$(J);CH$;T%(J);CH$;
317 GOSUB285:NEXTJ:CLOSE10:RETURN
319 R$="":GETR$:IFR$=""THEN319
321 RETURN
323 REM INIZIALIZZAZIONE DISCO
325 REM **********************
327 PRINT"NOME DISCO";:INPUTN$
329 PRINT#15,"N0:"+N$+",99"
331 CLOSE15:OPEN15,8,15:PRINT#15,"I"
333 RETURN
335 REM NUOVI DATI
337 REM **********
339 OPEN11,8,11,NN$:GOSUB285
341 GOSUB151
343 IFW=1THENK=K-1:CLOSE11:GOTO561
345 GOSUB235:IFJJ=0THENSTOP:REM STOP IMPOSS.
347 T=T%(JJ):GOSUB209:GOSUB193
349 REM AGGIORNAMENTO INDICE
351 REM ********************
353 C$(JJ)=Y$(1)+Y$(2)
355 K=K+1
357 IFK>NTHENK=K-1:PRINTS2$:CLOSE11:GOTO561
359 GOTO341
361 REM AGGIORNAMENTO
363 REM *************
365 PRINTTAB(10);"██AGGIORNAMENTO ARCHIVIO"
367 PRINT:PRINTTAB(10);"1=CORREZIONE"
369 PRINTTAB(10);"2=AGGIUNTA ELEM."
371 PRINTTAB(10);"3=CANCELL.ELEM."
373 PRINTTAB(10);"9=FINE"
375 INPUT"COSA SCEGLI ";X:IFX=9THEN561
377 IFX<1ORX>3THEN365
379 IFX=2THEN461
381 IFX=3THEN475
383 GOTO417
385 REM RICERCA RECORD
387 REM **************
389 PRINT"█";D$(1);:Y$(1)="":INPUTY$(1):W=0
391 IFY$(1)="$"THENW=1:RETURN
393 PRINTD$(2);:Y$(2)="":INPUTY$(2)
395 Y$(1)=LEFT$(Y$(1)+SP$,L(1))
397 Y$(2)=LEFT$(Y$(2)+SP$,L(2))
399 INPUT"QUALE OCCORRENZA ";X
401 IFX<=0THENPRINT"█";:GOTO399
403 FORJ=1TOK:IFC$(J)=Y$(1)+Y$(2)THEN411
405 NEXTJ
407 J=J-1:PRINT"NON TROVATO ";Y$(1);"  ";Y$(2)
409 GOSUB319:GOTO385
411 IFX<>1THENX=X-1:GOTO405
413 T=T%(J)
415 I=J:J=K:NEXTJ:RETURN
417 REM CORREZIONE
419 REM **********
421 GOSUB385:IFW=1THEN365
423 GOSUB247
425 REM VA A MODIFICA DATI
```

```
427 REM ******************
429 GOSUB439
431 PRINT#15,"I":OPEN11,8,11,NN$
433 GOSUB209:GOSUB193
435 C$(I)=Y$(1)+Y$(2)
437 CLOSE11:GOTO417
439 REM CONTROLLO DATI E MODIFICHE
441 REM ***************************
443 PRINT"□CONTROLLO DATI E MODIFICHE"
445 FORJ=1TONC:PRINTJ;" ";D$(J);" ";Y$(J):NEXTJ
447 INPUT"□TUTTO BENE S/N ";X$:IFX$="S"THENRETURN
449 PRINT"QUALE CAMPO (0 USCITA)";:INPUTX
451 IFX=0THENRETURN
453 IFX>NCORX<1THENPRINT"□□":GOTO447
455 Y$(X)=""
457 INPUTY$(X):Y$(X)=LEFT$(Y$(X)+SP$,L(X))
459 GOTO443
461 REM AGGIUNTA
463 REM**********
465 PRINT"□AGGIUNTA NUOVI RECORD"
467 PRINT"PRESENTI ";K;" RECORD"
469 PRINT"PUOI AGGIUNGERE ";N-K;" RECORD"
471 GOSUB319
473 W=0:K=K+1:GOTO335
475 REM CANCELLAZIONE
477 REM *************
479 M=0
481 GOSUB385:IFW=1THEN487
483 X=L(1)+L(2):C$(I)=LEFT$(LIM$+SP$+SP$,X)
485 M=M+1:GOTO481
487 REM SIST. INDICE
489 GOSUB581:K=K-M:GOTO573
491 REM LISTA FILE
493 REM **********
495 T$="LISTA PER INDICE PRIM."
497 PRINT"ACCENDI STAMPANTE      PREMI UN TASTO"
499 GETR$:IFR$=""THEN499
501 PRINT"□";T$:OPEN4,4
503 PRINT#4:PRINT#4:PRINT#4,T$
505 FORJ=1TO10:PRINT#4:NEXTJ
507 L=2:FORI=1TOK:T=T%(I):GOSUB247
509 PRINT#4,Y$(1);"  ";Y$(2)
511 PRINT#4,Y$(3);"  ";Y$(4);"  ";Y$(5)
513 FORM=6TONC:PRINT#4,D$(M);Y$(M):NEXTM
515 PRINT#4:PRINT#4
517 IFL=5THENPRINT#4:L=0
519 L=L+1:NEXTI
521 CLOSE4:GOTO575
523 REM CREAZIONE INDICE SECONDARIO
525 REM ***************************
527 PRINT"□INDICE SECONDARIO"
529 INPUT"□QUALE CAMPO ";X$
531 X=VAL(X$):IFX<2ORX>NCTHENGOTO529
533 FORI=1TOK:T=T%(I):GOSUB247
534 IFLEN(Y$(X))=0THENY$(X)=CHR$(160)
535 C$(I)=Y$(X):NEXTI
537 GOSUB305:GOSUB273
539 PRINT"FINITO IND. SEC."
541 GOTO575
543 REM LISTA PER INDICE SECONDARIO
545 REM ***************************
547 PRINT"□LISTA IND. SEC."
549 OPEN10,8,10,"INDI2,S,R":GOSUB285
551 FORJ=1TOK:INPUT#10,C$(J),T%(J)
```

```
553 GOSUB285:NEXTJ
555 CLOSE10:GOSUB607:GOSUB305
557 T$="LISTA IND. SEC. "
559 GOTO497
561 REM CHIUSURA
563 REM ********
565 PRINT"CHIUSURA ARCHIVIO"
567 PRINT"[illegible] ORDINA INDICE INDI1"
569 PRINT" ATTENDI CON PAZIENZA"
571 GOSUB581
573 GOSUB177:GOSUB221
575 PRINT"FINITO AGGIORNAMENTO"
577 PRINT"SONO PRESENTI ";K;" RECORD"
579 CLOSE15:STOP
581 REM ORDINAMENTO CRESCENTE
583 REM *********************
585 L=N-1
587 W=0
589 FORJ=1TOL
591 IFC$(J)<=C$(J+1)THEN599
593 R$=C$(J):C$(J)=C$(J+1):C$(J+1)=R$
595 X=T%(J):T%(J)=T%(J+1):T%(J+1)=X
597 W=1
599 NEXTJ
601 IFW=0THENRETURN
603 IFL=1THENRETURN
605 L=L-1:GOTO587
607 REM ORDINAMENTO DECRESCENTE
609 REM ***********************
611 L=K-1
613 W=0
615 FORJ=1TOL
617 IFC$(J)>=C$(J+1)THEN625
619 R$=C$(J):C$(J)=C$(J+1):C$(J+1)=R$
621 X=T%(J):T%(J)=T%(J+1):T%(J+1)=X
623 W=1
625 NEXTJ
627 IFW=0THENRETURN
629 IFL=1THENRETURN
631 L=L-1:GOTO613
```

Non riportiamo per esteso il commento al programma, ma segnaliamo le parti che riguardano il trattamento del file relativo.

. 111/137: preestensione file nella fase 1 del menu'.

. 193/207: viene preparata la stringa di scrittura di un record logico con tutti i campi.

. 209/219: definizione posizione record e puntatore.

. 235/245: ricerca posto nell'indice.

Anche per questo archivio devi fare delle considerazioni sull'occupazione di memoria e la quantita' di record trattabili, come per il file random.

A questo punto pensiamo che lo studio dei diversi programmi di archivio presentati ti potra' garantire una buona conoscenza dei diversi tipi di file gestibili e potrai, o modificare i programmi per adattarli alle tue esigenze, o prepararne altri, magari servendoti di alcune delle routine da noi utilizzate.
In questo capitolo non ci siamo preoccupati di presentare quadri video particolarmente belli, potrai migliorare tu le situazioni, usando anche i colori.
Inoltre, se vuoi, potrai controllare l'ingresso dati, con apposite routine, per evitare segnalazioni di errore che deturpano i quadri video.

3.8 PROGRAMMI DI UTILITA'

Si definiscono tali i programmi che sono di aiuto durante il lavoro di programmazione. Nella scatola dell'unita' 1541 si trova anche il dischetto TEST/DEMO, che contiene alcuni programmi di utilita'.
Riportiamo un breve commento ad alcuni di essi.
C-64 WEDGE carica il DOS 5.1; dopo averlo fatto girare sono attivi i seguenti comandi:
. /nome-programma, per caricare un programma dal dischetto.
. > oppure @, per visualizzare lo stato delle 4 variabili di errore disco.
. >$ oppure @$, per visualizzare la directory sul video, senza cancellare il programma presente in memoria.
Il programma resta attivo fino a quando togli corrente.

COPY/ALL ti permette di eseguire la copia dei dischetti collegando 2 unita' 1541. Devi cambiare il "dn" di una di esse da 8 a 9, con il programma DISK ADDR CHANGE. Il cambio di unita' resta valido fino a quando togli corrente. Prima di eseguire una copia conviene proteggere il disco sorgente chiudendo la finestrella laterale.
In commercio esistono programmi che consentono di eseguire la copia dei floppy anche disponendo di una sola unita'; viene richiesto con un messaggio di scambiare i floppy quando e' necessario.

DIR serve per: leggere e listare la directory, eseguire comandi del DOS, visualizzare le 4 variabili di errore. Il programma presenta un certo interesse per

le tecniche di programmazione usate. La directory viene aperta come file "$0" sul canale 0, con lfn=1 e poi letta con GET#1, carattere per carattere.

VIEW BAM visualizza sul video la BAM in due quadri video, come una matrice, riportando sull'asse orizzontale le tracce e sull'asse verticale i settori. Il programma contiene un'imperfezione, infatti nelle tracce da 18 a 24 sono mostrati senza l'annullamento N/L, ma come occupati, i settori 19 che non esistono.

CHECK DISK controlla se il dischetto ha settori rovinati, ma risulta molto lento, dopo aver segnalato eventuali settori rovinati lascia il dischetto tutto occupato. Andrebbe eseguito il comando COLLECT o VALIDATE e bisognerebbe escludere i settori rovinati marcandoli come occupati sia nella BAM che nella directory.

DISPLAY T&S serve per visualizzare o su video o su stampante il contenuto di uno o piu' settori del floppy. Vengono mostrati a sinistra i contenuti esadecimali e a destra i caratteri di stampa. In certi casi va in crisi; puoi premere RESET mentre tieni premuto RUN/STOP, poi uscire dal MONITOR con X e ripartire.

Riportiamo ora alcuni programmi di utilita' preparati da noi.

DCOMEFS stampa la BAM e la directory. Nella prima parte vengono evidenziati i 35 gruppi di 4 byte dedicati alla BAM, vedi descrizione del Paragrafo 3.1. Dopo sono stampate le informazioni relative ad ogni entrata della directory.
Il programma manda l'uscita alla stampante, puoi dirigerla al video modificando la linea 30 in OPEN4,3.

```
5 REM DCOMEFS
10 T$(0)="DEL":T$(1)="SEQ":T$(2)="PRG"
15 T$(3)="USR":T$(4)="REL"
20 CLOSE15:OPEN15,8,15,"I"
25 OPEN2,8,2,"$":REM APERTURA FILE DIRECTORY
30 OPEN4,4:REM APERTURA STAMPANTE
35 PRINT#4,"STAMPA DIRECTORY COME FILE SEQUENZIALE"
40 PRINT#4
45 REM LETTURA PRIMI 2 BYTE
```

```
50 I=2:GOSUB230
55 REM LETTURA BAM
60 FORK=1 TO 35:PRINT#4,K;"    ";
65 I=4:GOSUB230
70 NEXTK:PRINT#4
75 REM LETTURA NOME DISCO
80 I=18:GOSUB205
85 PRINT#4,N$;"   ";
90 REM LETTURA ID
95 I=2:GOSUB205:PRINT#4,N$
100 REM LETTURA ALTRI 7 BYTE
105 I=7:GOSUB230
110 PRINT#4
115 REM LETTURA ALTRI 85 BYTE
120 FORK=1TO85:GET#2,A$:NEXTK
125 PRINT#4
130 FORJ=1TO8
135 GET#2,T$,A$,B$
140 IFSTTHENCLOSE2:CLOSE4:CLOSE15:STOP
145 IFT$=""THENT$=CHR$(128)
150 I=16
155 GOSUB205
160 PRINT#4,T$(ASC(T$)-128);"   ";
165 PRINT#4,ASC(A$+CHR$(0));ASC(B$+CHR$(0));
170 PRINT#4,N$
175 I=3:GOSUB230
180 FORK=1TO4:GET#2,A$:NEXTK
185 I=2:GOSUB230
190 I=2:GOSUB230
195 IFJ<8THENGET#2,A$,A$
200 NEXTJ:GOTO130
205 REM COSTRUZIONE STRINGA
210 N$="":FORL=1TOI
215 GET#2,L$
220 IFL$<>CHR$(96)THENIFL$<>CHR$(160)THENN$=N$+L$
225 NEXTL:RETURN
230 REM LETTURA E STAMPA GRUPPO BYTE
235 FORL=1TOI
240 GET#2,A$:PRINT#4,ASC(A$+CHR$(0));
245 NEXTL:PRINT#4
250 RETURN
```

CONTRDISCO controlla se sono registrati bene sul disco i file di tipo SEQ e PRG, ma non controlla i file di altro tipo. Ogni volta che controlla un blocco fa apparire un pallino sul video. Se un file non e' in ordine ti chiede se vuoi cancellarlo. Se rispondi S, lo cancella e esegue il comando VALIDATE, se rispondi N si ferma.

```
5 REM CONTRDISCO
10 REM --COSTANTI E VARIABILI--
15 M1$="**************":FI=0
20 M2$="▮FILE NON CORRETTO: ▮"
25 M3$="▮TUTTO IN ORDINE   ...▮ SPERO !"
30 REM --APRE CANALE 15 CON LFN=1--
```

```
35 OPEN1,8,15,"I"
40 REM --APRE CANALE 2 PER DIRECTORY--
45 OPEN2,8,2,"#"
50 REM --APRE CANALE 3 PER FILE--
55 OPEN3,8,3,"#"
60 REM --FUNZ. PER PUNTARE ENTRATE DIRECTORY--
65 DEFFNF(X)=2+32*X
70 REM --PRIMO BLOCCO DIRECTORY--
75 TD=18:SD=1:GOSUB350
80 REM --ELABORA UNA ENTRATA--
85 FI$="":CB=0
90 REM --PUNTATORE A INIZIO ENTRATA--
95 PRINT#1,"B-P:2,"FNF(FI)
100 REM --LEGGE TIPO FILE CHE VA IN TY--
105 GET#2,A$:GOSUB385:TY=ASC(A$)
110 REM --LEGGE IND. INIZIO FILE, TR E SC--
115 GET#2,A$,B$:GOSUB380:TR=ASC(A$):SC=ASC(B$)
120 REM --LEGGE NOME FILE--
125 FORI=1TO16:GET#2,A$:FI$=FI$+A$
130 IFA$=CHR$(160)THENFI$=LEFT$(FI$,I-1):I=16
135 NEXT
140 REM --PUNTATORE AI 2 BYTE CONT. BLOCCHI--
145 PRINT#1,"B-P:2,"FNF(FI)+28:GET#2,A$,B$
150 REM --NB=VCONTATORE BLOCCHI--
155 GOSUB380:NB=ASC(A$)+ASC(B$)*256
160 REM --CONTROLLO TIPO FILE 1 O 2--
165 IFTY<>0THENTY=TY-128
170 IFTY<>2ANDTY<>1THEN310
175 IFTY<>2ANDTY<>1THEN85
180 REM --CONTROLLO BLOCCHI FILE--
185 PRINT"♥QQQ"M1$
190 PRINT"CONTROLLO FILE":PRINTM1$"QQQ"
195 PRINTTAB(10)FI$"QQ"
200 REM --LEGGE BLOCCO FILE--
205 PRINT#1,"B-R:3,0,"TR,SC
210 REM --LEGGE BYTE CONCATENAMENTO--
215 PRINT#1,"B-P:3,0"
220 CB=CB+1:GET#3,A$,B$:GOSUB380
225 REM --STAMPA UN PALLINO PER OGNI BLOCCO--
230 TR=ASC(A$):SC=ASC(B$):PRINT"●";
235 IFTR<>0THEN205
240 REM --CONTROLLO LUNGHEZZA FILE--
245 IFCB=NBTHEN325
250 REM --CICLO DI ATTESA--
255 IFCB=NBTHENFORI=1TO250:NEXTI:GOTO85
260 PRINT:PRINTM2$FI$
265 REM --SE CONTEGGIO ERRATO--
270 PRINT"QDEVO AZZERARE  ? (RS■/RN■)"
275 GETA$:IFA$=""THEN275
280 IFA$="N"THEN400
285 IFA$<>"S"THEN275
290 PRINT"QAZZERAMENTO E VALIDATE"
295 PRINT#1,"S:"FI$:PRINT#1,"V"
300 CLOSE3:CLOSE2:CLOSE1:RUN
305 REM --PASSA A PROSSIMA ENTRATA--
310 FI=FI+1:IFFI=8THENGOSUB350:FI=0
315 GOTO175
320 REM --TUTTO BENE--
325 PRINT:PRINTTAB(9)"QQQQ]OK":FI=FI+1
330 IFFI=8THENGOSUB350:FI=0
335 GOTO255
340 REM --RE LETTURA DIRECTORY--
345 REM --TD=0 A FINE DIRECTORY--
350 IFTD=0THENPRINTM3$:GOTO400
```

```
355 REM --LETTURA BLOCCO--
360 A$="":B$="":PRINT#1,"U1:2,0";TD;SD
365 GET#2,A$,B$:GOSUB380
370 TD=ASC(A$):SD=ASC(B$):RETURN
375 REM --SISTEMAZIONE BYTE LETTI--
380 IFB$=""THENB$=CHR$(0)
385 IFA$=""THENA$=CHR$(0)
390 RETURN
395 REM --CHIUSURA FILE--
400 CLOSE3:CLOSE2:CLOSE1:STOP
```

TRAC/SET e' una modifica di DISPLAY T&S e consente di visualizzare gruppi di settori concatenati e settori isolati.

```
5 REM TRAC/SET
10 RC$="        █V█IDEO ███████████D   ███P█RINTER"
15 RD$="        ██VIDEO █"
20 RE$="        ██PRINTER█"
25 PRINT"███________________________________"
30 PRINT"  CONTENUTO BLOCCHI    "
35 PRINT"______________________________"
40 REM COSTANTI
45 SP$=" ":NL$=CHR$(0)
50 HX$="0123456789ABCDEF"
55 FS$="":FORI=64 TO 95
60 FS$=FS$+"█"+CHR$(I)+"█":NEXT I
65 SS$="  ":FOR I=192 TO 223
70 SS$=SS$+"█"+CHR$(I)+"█":NEXT I
75 DIM A$(15),NB(2)
80 D$="0"
85 PRINTRC$
90 GETJJ$:IF JJ$="" THEN90
95 IF JJ$="V"THENPRINTRD$
100 IF JJ$="P"THENPRINTRE$:OPEN4,4
110 GOSUB450
115 REM CARICA BUFFER
120 INPUT"███TRACCIA, SETTORE";T,S
125 IF T=0 OR T>35 THEN355
130 IF JJ$="V" THENGOSUB360:GOTO135
131 PRINT#4:PRINT#4,"TRACCIA"T" SETTORE"S:PRINT#4
135 PRINT#15,"U1:2,"D$;T;S:GOSUB335
140 PRINT#15,"B-P:2,0"
145 REM LEGGE BYTE 0
150 GET#2,A$(0):IFA$(0)=""THENA$(0)=NL$
155 PRINT#15,"B-P:2,1"
160 IF JJ$="V"THEN170
165 IF JJ$="P"THEN230
170 REM VIDEO
175 K=1:NB(1)=ASC(A$(0))
180 FORJ=0TO63:IFJ=32THENGOSUB375:GOTO365
185 FOR I=K TO 3
190 GET#2,A$(I):IF A$(I)="" THEN A$(I)=NL$
195 IF K=1 AND I<2 THEN NB(2)=ASC(A$(I))
200 NEXT I:K=0
```

```
205 A$="":B$=":":N=J*4:GOSUB 405:A$=A$+":"
210 FOR I=0 TO 3:N=ASC(A$(I)):GOSUB 405
215 C$=A$(I):GOSUB 425:B$=B$+C$
220 NEXT I:IF JJ$="V" THEN PRINTA$B$
225 NEXT J:GOTO295
230 REM PRINTER
235 K=1:NB(1)=ASC(A$(0))
240 FOR J=0 TO 15
245 FOR I=K TO 15
250 GET#2,A$(I):IF A$(I)="" THEN A$(I)=NL$
255 IF K=1 AND I<2 THEN NB(2)=ASC(A$(I))
260 NEXT I:K=0
265 A$="":B$=":":N=J*16:GOSUB 405:A$=A$+":"
270 FOR I=0 TO 15:N=ASC(A$(I)):GOSUB 405
275 C$=A$(I):GOSUB 425:B$=B$+C$
280 NEXT I
285 IF JJ$="P" THEN PRINT#4,A$B$
290 NEXT J:GOTO295
295 REM SUCCESSIVO BLOCCO
300 PRINT"TRACCIA/SETT.SEGUENTE"NB(1)NB(2) "■"
305 PRINT"DESIDERI PROSEGUIRE          S/N   "
310 GET Z$:IF Z$="" THEN310
315 IF Z$<>"S" THEN325
320 T=NB(1):S=NB(2):GOSUB445:GOSUB450:GOTO125
325 IF Z$="N" THEN GOSUB445:GOSUB450:GOTO120
330 GOTO 310
335 REM ROUTINE ERRORE
340 INPUT#15,EN,EM$,ET,ES:IF EN=0 THEN RETURN
345 PRINT"■ERRORE DISCO■"EN,EM$,ET,ES
350 END
355 PRINT#15,"I"D$:GOSUB445:CLOSE4:PRINT"END":END
360 PRINT"■■■TRACCIA"T" SETTORE"S"■":RETURN
365 IF Z$="N"THEN J=80:GOTO 225
370 GOTO185
375 REM MESS. CONTINUAZIONE
380 PRINT"■■■■■CONTINUO(S/N)"
385 GETZ$:IF Z$="" THEN 385
390 IF Z$="N" THEN RETURN
395 IF Z$<>"S" THEN 385
400 PRINT"■TRACCIA" T " SETTORE"S:RETURN
405 REM CONV.HEX
410 A1=INT(N/16):A$=A$+MID$(HX$,A1+1,1)
415 A2=INT(N-16*A1):A$=A$+MID$(HX$,A2+1,1)
420 A$=A$+SP$:RETURN
425 REMCONV.ASCII
430 IF ASC(C$)<32 THEN C$=" ":RETURN
435 IF ASC(C$)<128 OR ASC(C$)>159 THEN RETURN
440 C$=MID$(SS$,3*(ASC(C$)-127),3):RETURN
445 CLOSE2:CLOSE15:RETURN
450 OPEN15,8,15,"I"+D$:GOSUB335
455 OPEN2,8,2,"#":GOSUB335:RETURN
```

SIST/DISCO controlla il dischetto per trovare se ci sono settori rovinati; esso e' una modifica di CHECK DISK, ma risulta molto piu' veloce, infatti abbiamo ottimizzato l'allocazione dei blocchi. Alla fine il dischetto risulta con allocati nella BAM i settori rovinati, ma non nella directory, quindi puo' essere usato, ma non puoi eseguire i comandi VALIDATE o COLLECT.

```
5 REM SIST/DISCO
10 REM TABELLA TRACCE E SETTORI GUASTI
15 DIMT(100):DIMS(100)
20 DIMG1(21),G2(19),G3(18),G4(17)
25 DATA0,10,20,8,18,6,16,4,14,2,12,9,19
30 DATA7,17,5,15,3,13,1,11
35 DATA0,11,1,12,2,13,3,14,4,15,5,16,6,17
40 DATA7,18,8,9,10
45 DATA0,10,1,11,2,12,3,13,4,14,5,15
50 DATA6,16,7,17,8,9
55 DATA0,10,1,11,2,12,3,13,4,14,5
60 DATA15,6,16,7,8,9
65 FORJ=1TO21:READG1(J):NEXTJ
70 FORJ=1TO19:READG2(J):NEXTJ
75 FORJ=1TO18:READG3(J):NEXTJ
80 FORJ=1TO17:READG4(J):NEXTJ
85 RC$=" BLOCCHI GUASTI SONO STATI ALLOCATI"
90 RD$="BLOCCHI GUASTI":RE$="TRACCIA"
95 RF$="SETTORE"
100 PRINT"────────────────────────────"
105 PRINT"      SIST/DISCO        "
110 PRINT"────────────────────────────"
115 OPEN15,8,15
120 PRINT#15,"V0"
125 N%=RND(TI)*255
130 A$="":FORI=1TO255
135 A$=A$+CHR$(255AND(I+N%)):NEXT
140 OPEN2,8,2,"#":GOSUB325
145 PRINT:PRINT#2,A$;
150 J=1
155 FORT=1TO17:FORL=1TO21
160 S=G1(L):GOSUB200:NEXTL:NEXTT
165 FORT=18TO24:FORL=1TO19
170 S=G2(L):GOSUB200:NEXTL:NEXTT
175 FORT=25TO30:FORL=1TO18
180 S=G3(L):GOSUB200:NEXTL:NEXTT
185 FORT=31TO35:FORL=1TO17
190 S=G4(L):GOSUB200:NEXTL:NEXTT
195 GOTO260
200 PRINT#15,"B-A:0"T;S
205 INPUT#15,EN,EM$,ET,ES
210 IFEN=0THEN225
215 IFEN=65THENRETURN
220 STOP
225 PRINT#15,"U2:2,0"T;S
230 NB=NB+1
235 INPUT#15,EN,EM$,ES,ET
240 IF EN=0THENRETURN
245 T(J)=T:S(J)=S:J=J+1
250 PRINT"BLOCCHI GUASTI:";T;S
255 RETURN
260 PRINT#15,"V0"
265 GOSUB325
270 CLOSE2
275 IFJ=1THEN350
280 OPEN2,8,2,"#"
285 PRINTRD$,RE$,RF$
295 FORI=1TOJ-1
300 PRINT#15,"B-A:0"T(I);S(I)
305 PRINT,,T(I),S(I)
310 NEXT
315 PRINT"";J-1;RC$
320 CLOSE2:CLOSE15:END
325 INPUT#15,EN,EM$,ET,ES
```

```
330 IF EN=0 THEN RETURN
335 PRINT"ERRORE#"EN,EM$;ET;ES""
340 PRINT#15,"I0"
345 RETURN
350 PRINT"TUTTO BENE!    ":CLOSE15:END
```

NUMBUFFER serve per trovare quanti buffer dell'unita' 1541 possono essere aperti contemporaneamente in modo diretto e quali sono i loro numeri.

```
5 REM NUMBUFFER
10 OPEN15,8,15,"I"
15 INPUT"QUANTI BUFFER";N
20 A$="":B$="":C$="":D$="":E$=""
25 ONNGOTO50,45,40,35,30
30 OPEN2,8,2,"#":GOSUB115:GET#2,E$:GOSUB115
35 OPEN3,8,3,"#":GOSUB115:GET#3,D$:GOSUB115
40 OPEN4,8,4,"#":GOSUB115:GET#4,C$:GOSUB115
45 OPEN5,8,5,"#":GOSUB115:GET#5,B$:GOSUB115
50 OPEN6,8,6,"#":GOSUB115:GET#6,A$:GOSUB115
55 ONNGOTO100,90,80,70,60
60 IFE$=""THENE$=CHR$(0)
65 PRINTASC(E$):CLOSE2
70 IFD$=""THEND$=CHR$(0)
75 PRINTASC(D$):CLOSE3
80 IFC$=""THENC$=CHR$(0)
85 PRINTASC(C$):CLOSE4
90 IFB$=""THENB$=CHR$(0)
95 PRINTASC(B$):CLOSE5
100 IFA$=""THENA$=CHR$(0)
105 PRINTASC(A$):CLOSE6
110 CLOSE15:STOP
115 INPUT#15,EN,EM$,ET,ES
120 PRINTEN;EM$;ET;ES
125 RETURN
```

INDBUFFER consente di vedere quali indirizzi vengono assegnati ad ogni buffer nella RAM dell'unita' 1541.

```
5 REM INDBUFFER
10 DIMR$(8,7):FORK=1TO8:FORI=1TO7
15 R$(K,I)="":NEXTI:NEXTK
16 M1$="IND. IN MEMORIA DEL BUFFER: "
```

```
20 OPEN7,4:PRINT#7,"RISULTATI INDBUFFER"
25 OPEN15,8,15,"I"
30 INPUT"QUANTI BUFFER";N:PRINT#7
35 PRINT#7,N;" BUFFER":PRINT#7
40 A$="":B$="":C$="":D$="":E$=""
45 ONNGOTO70,65,60,55,50
50 OPEN2,8,2,"#":GOSUB270:GET#2,E$:GOSUB270
55 OPEN3,8,3,"#":GOSUB270:GET#3,D$:GOSUB270
60 OPEN4,8,4,"#":GOSUB270:GET#4,C$:GOSUB270
65 OPEN5,8,5,"#":GOSUB270:GET#5,B$:GOSUB270
70 OPEN6,8,6,"#":GOSUB270:GET#6,A$:GOSUB270
75 PRINT#7,"NUMERI DEI BUFFER: ";
80 ONNGOTO165,145,125,105,85
85 IFE$=""THENE$=CHR$(0)
90 PRINT#7,ASC(E$);
95 PRINT#15,"B-P:"2;1
100 PRINT#2,"22222";:GOSUB270
105 IFD$=""THEND$=CHR$(0)
110 PRINT#7,ASC(D$);
115 PRINT#15,"B-P:"3;1
120 PRINT#3,"33333";:GOSUB270
125 IFC$=""THENC$=CHR$(0)
130 PRINT#7,ASC(C$);
135 PRINT#15,"B-P:"4;1
140 PRINT#4,"44444";:GOSUB270
145 IFB$=""THENB$=CHR$(0)
150 PRINT#7,ASC(B$);
155 PRINT#15,"B-P:"5;1
160 PRINT#5,"55555";:GOSUB270
165 IFA$=""THENA$=CHR$(0)
170 PRINT#7,ASC(A$);
175 PRINT#15,"B-P:"6;1
180 PRINT#6,"66666";:GOSUB270
185 PRINT#7
190 FORK=1TO8:M=K-1
195 FORI=1TO7:L=I-1
200 PRINT#15,"M-R"CHR$(L)CHR$(M)
205 GET#15,R$(K,I)
210 NEXTI
215 NEXTK
220 FORK=1TO8:FORI=1TO7
225 IFR$(K,I)=""THENR$(K,I)=CHR$(0)
230 NEXTI:NEXTK
235 FORK=1TO8
240 PRINT#7,M1$;STR$((K-1)*256)
245 PRINT#7,"PRIMI 8 BYTE DEL BUFFER:";
250 FORI=1TO7
255 PRINT#7,R$(K,I);:NEXTI:PRINT#7
260 NEXTK
265 FORK=1TO7:CLOSEK:NEXTK:CLOSE15:STOP
270 INPUT#15,EN,EM$,ET,ES
275 PRINTEN;EM$;ET;ES
280 RETURN
```

CONTRINDI1 consente di vedere come il sistema assegna gli indirizzi di traccia e settore al file sequenziale INDI1, usato come indice degli archivi trattati nei precedenti paragrafi.

```
10 REM CONTRINDI1
20 OPEN15,8,15,"I"
30 OPEN2,8,2,"INDI1,S,R"
35 OPEN4,4:PRINT#4,"CONTROLLO INDI1":PRINT#4
40 K=0:I=0
45 INPUT#2,A$,T,S:TS=ST:K=K+1
50 PRINT#4,K;T;S;" ";
60 I=I+1:IFI=5THENPRINT#4:I=0
70 IFTS=64THENPRINT#4:CLOSE2:CLOSE4:CLOSE15:STOP
80 GOTO45
```

ALLOCA mostra come devono essere correttamente allocati i blocchi in modo diretto senza invadere la traccia 18, che viene usata per la directory.

```
1000 REM ALLOCA
1005 REM TRACCIA TX, SETTORE SX
1010 REM SI SUPPONE APERTO UN CANALE DATI
1015 REM SI SUPPONE APERTO IL CANALE 15
1020 REM SI SUPPONE CHE SX SIA COMPATIBILE CON TX
1025 REM SCARTA LA TRACCIA 18
1030 PRINT#15,"B-A:0";TS;SX
1035 INPUT#15,EN,EM$,ET,ES
1040 IFEN=0THEN RETURN
1045 IFEN<>65THEN STOP
1050 IFET=18THENTX=19:SX=0:GOTO1030
1055 TX=ET:SX=ES:GOTO1030
```

CAPITOLO 4

ARCHITETTURA DEL SISTEMA

4.1 INTRODUZIONE

In questo capitolo affrontiamo un argomento avanzato; vogliamo descrivere il funzionamento del calcolatore nelle sue singole parti, cercando di renderlo comprensibile anche a chi non ha mai applicato CIRCUITI INTEGRATI, e non ha conoscenze di ELETTRONICA DIGITALE. Finora abbiamo visto il calcolatore come un apparecchio che, quasi per magia, comprende un linguaggio che assomiglia un po' alla lingua inglese, il BASIC. Qui trattiamo gli argomenti che occorre conoscere per essere in grado di adoperare il LINGUAGGIO MACCHINA, la cui conoscenza e' istruttiva ed assieme utile per risolvere problemi di velocita' e di gestione diretta delle periferiche.
Il tuo COMMODORE 16 possiede al suo interno diversi circuiti integrati; dei piu' importanti cercheremo di dare una descrizione soddisfacente, ma consapevolmente non completa per non disperdere in dettagli l'argomento.
Il COMMODORE 16, come tutti i calcolatori, e' costituito da circuiti integrati, collegati tra loro mediante connessioni elettriche. Ogni circuito integrato e' costruito in modo da svolgere determinate funzioni; il costruttore generalmente fornisce, a richiesta, la documentazione necessaria per applicarlo. Nel COMMODORE 16 alcuni integrati sono stati costruiti apposta dalla COMMODORE, e questa soluzione ha permesso di ridurre notevolmente il numero di componenti necessari per il funzionamento del calcolatore. Le principali componenti di un calcolatore sono tre:
1). UNITA' CENTRALE DI ELABORAZIONE (CPU, che in inglese significa CENTRAL PROCESSING UNIT)
2). MEMORIA

3). DISPOSITIVI DI INGRESSO/USCITA (I/O, cioe' INPUT/OUTPUT)
La potenza di un calcolatore dipende dalla potenza di queste tre parti.

4.2 LA MEMORIA

LA MEMORIA e' concettualmente la piu' semplice delle tre parti. Esistono nel COMMODORE 16 due tipi di memoria: a sola lettura (ROM, Read Only Memory) e a lettura e scrittura (RAM). Come dice il nome stesso, la MEMORIA e' un dispositivo che ha la caratteristica di ricordare, e difatti ricorda i bit, che sono l'unita' di informazione. Come abbiamo visto nell'appendice B del libro "COMMODORE 16 per te", un BYTE di memoria e' l'insieme di 8 bit. La memoria e' organizzata in celle (byte): nel COMMODORE 16 ci sono 32768 celle di memoria ROM e 16384 celle di memoria RAM.
La memoria ROM e' stata costruita con i bit a un valore prefissato, in modo tale da non essere alterabile nel tempo. La ROM contiene tutto il SISTEMA OPERATIVO, cioe' l'insieme delle routine in linguaggio macchina che occorrono per comunicare con le periferiche (tastiera, video, floppy disk, registratore, stampante, ecc), e l'interprete BASIC, cioe' quel programma, scritto in linguaggio macchina, che, appoggiandosi alle routine del SISTEMA OPERATIVO, interpreta i comandi impartiti da tastiera, pone in memoria le linee introdotte, ed esegue le istruzioni memorizzate.
La memoria RAM ha la caratteristica di essere altera-

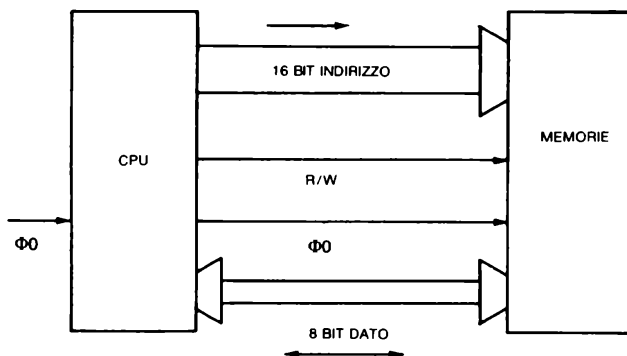



Figura 4.1 Collegamento tra CPU e memoria

bile: ogni byte di memoria RAM puo' cioe' essere scritto o letto, a seconda di un segnale, controllato dalla CPU (il segnale R/W, Read/Write). Quando questo segnale e' basso, l'accesso alla memoria viene effettuato in scrittura, altrimenti in lettura. La RAM e' usata per memorizzare i programmi dell'utente, le variabili, le informazioni da visualizzare sullo schermo ecc.
Nella Figura 4.1 e' illustrato il collegamento tra CPU e memoria. Per leggere un byte di memoria (vedi Figura 4.2) la CPU presenta sui 16 bit di indirizzo l'indirizzo del byte che vuole leggere, pone a 1 il segnale R/W, attende una transizione del segnale di temporizzazione (CLOCK) e legge il dato dagli 8 bit dedicati ai dati.

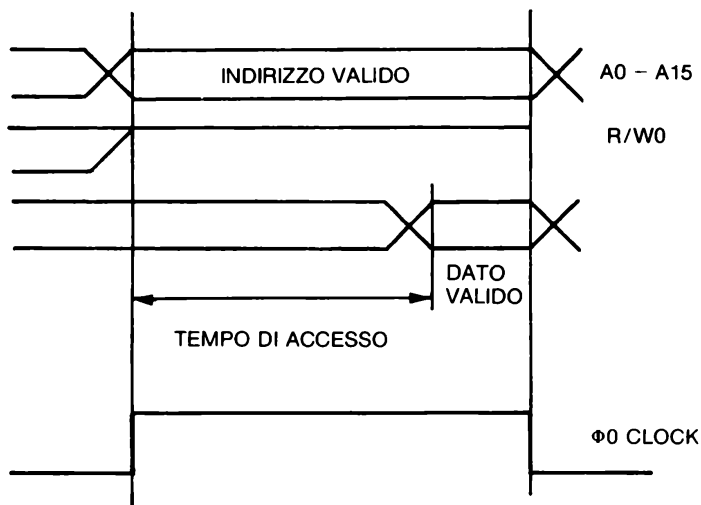



Figura 4.2 Ciclo di lettura da memoria

Per scrivere in memoria (vedi Figura 4.3) la CPU presenta sul DATA BUS (8 linee dedicate ai dati) il dato da scrivere, sul BUS INDIRIZZI (16 linee dedicate agli indirizzi) l'indirizzo del byte in cui memorizzare il dato e pone basso il segnale R/W. Dopo una transizione del segnale di CLOCK la CPU considera compiuta la scrittura.

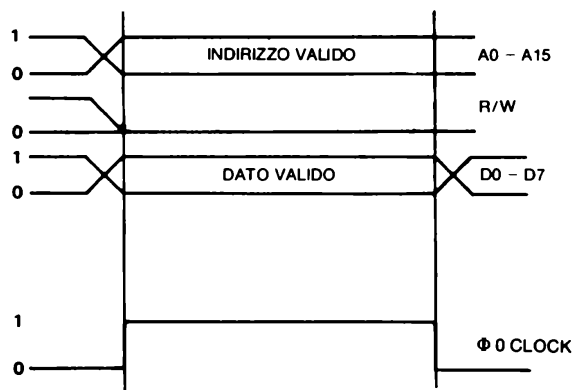



Figura 4.3 Ciclo di scrittura in memoria

4.3 L'UNITA' CENTRALE DI ELABORAZIONE (CPU)

La CPU e' considerata il CUORE del calcolatore, perche' esegue il programma, controlla i dispositivi di I/O e di fatto produce l'elaborazione dei dati secondo le istruzioni del programma. Nel COMMODORE 16 la CPU e' un circuito integrato con 40 piedini, cioe' 40 punti di collegamento con l'esterno. Questo integrato

Φoin	1	40	RES
RDY	2	39	R/$\overline{W}$
$\overline{IRQ}$	3	38	D0
AEC	4	37	D1
Vcc	5	36	D2
A0	6	35	D3
A1	7	34	D4
A2	8	33	D5
A3	9	32	D6
A4	10	31	D7
A5	11	30	P0
A6	12	29	P1
A7	13	28	P2
A8	14	27	P3
A9	15	26	P4
A10	16	25	P6
A11	17	24	P7
A12	18	23	GATE IN
A13	19	22	A15
GND	20	21	A14

Figura 4.4 Zoccolatura della CPU 7501

porta la sigla 7501, ed e' stato progettato apposta per i calcolatori COMMODORE 16 e PLUS 4. In realta' deriva da un'altra CPU, la famosa 6502, da cui la 7501 differisce per alcuni dettagli, ma della quale conserva la stessa architettura e lo stesso set di istruzioni.
Prima di vedere l'architettura di questa CPU descriviamo brevemente i 40 segnali che vi fanno capo.
Nella Figura 4.4 e' riportata la ZOCCOLATURA della CPU 7501.

. 1- Φ0 in. Ingresso. Da questo piedino la CPU riceve la temporizzazione (CLOCK) necessaria per il funzionamento. E' un' ONDA QUADRA della frequenza selezionabile di 890000 o 1780000 hertz. Tutte le operazioni della CPU sono legate al CLOCK: quando il CLOCK e' alto, la CPU accede a memoria, quando e' basso la CPU "pensa", cioe' esegue le commutazioni necessarie per eseguire l'istruzione, esegue calcoli, e non usa la memoria (vedi Figura 4.5).

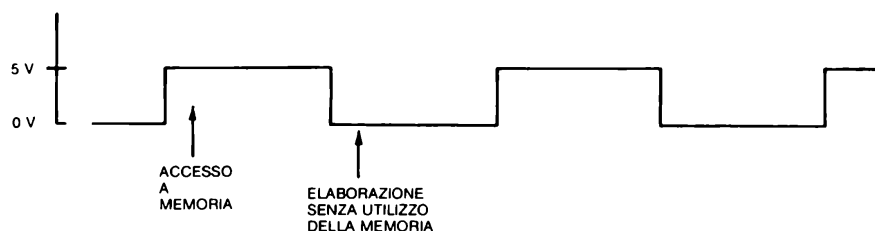



Figura 4.5 Segnale Φ0, il clock del sistema

.2- RDY. READY. Ingresso. Quando questo ingresso e' basso la CPU si arresta, e riparte quando RDY ritorna alto.

.3- IRQ. Interrupt ReQuest, ingresso. Quando si verifica una transizione da alto a basso di questo segnale la CPU "sente" una richiesta di interruzione, e va ad eseguire una speciale routine di servizio dell'interrupt (operazione simile alla chiamata e all'esecuzione di un sottoprogramma da programma, solo che qui la chiamata avviene per effetto di questo segnale), alla fine della quale il programma riprende.

.4- AEC. Address Enable Control, ingresso. Quando questo segnale viene posto basso la CPU non presenta gli indirizzi sul BUS INDIRIZZI. Cio' permette ad altri dispositivi di accedere alla memoria escludendo la CPU.

.5- VCC. Ingresso. Da questo piedino il circuito riceve l'alimentazione necessaria per funzionare. La tensione di alimentazione e' 5 volt in corrente continua, e la CPU assorbe circa 125 milliampere.
.6-19 e 21-22. ADDRESS BUS, uscite. Questi 16 piedini formano il BUS INDIRIZZI, cioe' la combinazione dei livelli logici di questi 16 bit individua uno tra i 2^16 (65536) possibili byte di memoria indirizzabili.
.20- GND, ingresso. E' il piedino di MASSA, da dove la CPU riceve il negativo di alimentazione, e rispetto al quale vanno misurati i livelli di tutti i segnali.
.23- GATE IN, ingresso.
.24-30. P0-P7, ingressi o uscite, programmabili. Questi 7 piedini fungono da I/O, e sono usati per collegare il calcolatore con il REGISTRATORE A CASSETTE, il FLOPPY DISK e la STAMPANTE.
.31-38 DATA BUS, ingressi e uscite. Su questi 8 BIT vengono scambiate le informazioni tra CPU e MEMORIA. Durante un ciclo di scrittura, sono uscite per la CPU, e ingressi per le memorie, mentre in un ciclo di lettura dalla memoria queste linee diventano uscite per la memoria e ingresso per la CPU.
.39- R/W, Read/Write, uscita. Questo segnale indica se il ciclo corrente di accesso alla memoria deve essere effettuato in lettura (livello ALTO) o in scrittura (livello BASSO).
.40 RES, ingresso, RESet. Questa linea e' sempre alta, durante il funzionamento del calcolatore. Quando viene posta bassa, la CPU arresta il programma in corso, e va ad eseguire la routine di inizializzazione. Il pulsante grigio sul fianco del COMMODORE 16 e' collegato alla linea di RESET. La linea di RESET viene posta bassa automaticamente all'accensione del calcolatore per circa un secondo.

4.4 I DISPOSITIVI DI INGRESSO/USCITA (I/O)

I dispositivi di INGRESSO/USCITA sono tutti quei dispositivi che collegano la CPU con l'esterno. La CPU piu' veloce del mondo, dotata della memoria piu' capace del mondo, sarebbe del tutto inutilizzabile se non fosse collegata ad almeno un dispositivo di ingresso e a un dispositivo di uscita. Immagina il tuo COMMODORE 16 privo di tastiera, video, suono, registratore a cassetta, stampante, floppy disk e di tutti gli altri

dispositivi di I/O: non servirebbe a niente. Nei paragrafi che seguono non parliamo dei dispositivi fisici che sono o possono essere attaccati al calcolatore, ma vediamo come funziona il collegamento tra CPU e dispositivi. Premettiamo che la CPU 7501 non prevede un collegamento particolare con i dispositivi di I/O; per questo tali dispositivi vengono collegati in modo da apparire come celle di memoria. Una porta di I/O e' come una cella di memoria "aperta" (con i bit collegati a piedini), che puo' essere usata in ingresso o in uscita, il cui stato puo' essere letto (in ingresso) come un normale byte di memoria, o puo' apparire (in uscita) ad altre parti del calcolatore. Generalmente una porta di I/O ha 8 bit, cosi' come sono 8 i bit di un byte di memoria.

4.5 LA TASTIERA

Nella tastiera del COMMODORE 16 vi sono 66 tasti, ma poiche' i due tasti SHIFT e SHIFT LOCK sono collegati in parallelo, vi sono 64 tasti indipendenti. Si potrebbe pensare che vi siano 8 porte di I/O da 8 bit, cioe' che ognuno dei 64 tasti sia collegato a 1 bit di

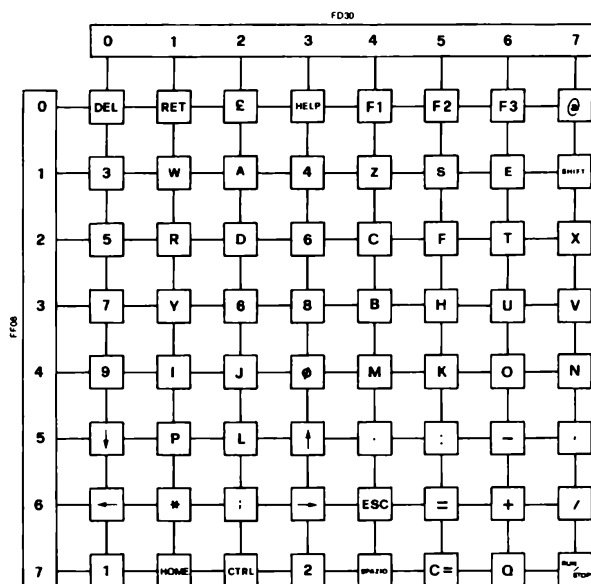



Figura 4.6 Organizzazione della tastiera

ingresso. Questa soluzione non sarebbe stata la piu' economica, e percio' la tastiera e' stata organizzata in una matrice di 8 righe e 8 colonne, in cui i tasti collegano i fili orizzontali con quelli verticali (vedi Figura 4.6); una porta di uscita e' collegata alle righe, una porta di ingresso alle colonne.
Per vedere se un tasto di una riga e' premuto, il SISTEMA OPERATIVO pone a 0 il bit della riga a cui quel tasto appartiene, a 1 i bit di tutte le altre righe, e legge dalla porta di ingresso lo stato delle colonne. Se qualcuno dei bit letti si trova a 0 allora vuol dire che il tasto corrispondente e' stato premuto, mentre gli ingressi delle colonne dove non sono premuti tasti vengono trovati nello stato logico 1. Il programma che segue, TASTMAT, si appoggia al SISTEMA OPERATIVO e mostra come la tastiera sia organizzata a matrice: la matrice disegnata ha 8 righe e 8 colonne: gli elementi della matrice sono quindi 64, e rappresentano ciascuno un tasto. Tenendone premuto uno vedi che un elemento della matrice cambia, e appare uno 0 al posto di un 1: nel programma infatti abbiamo rappresentato con 0 i tasti premuti e con 1 i tasti non premuti.

```
0 REM TASTMAT
10 SCNCLR:PRINT"   01234567":PRINT
20 DIMD(7):FORI=0TO7:D(I)=2↑I:NEXT
30 DO:FORI=0TO7
40 POKE2034,255-D(I):SYS56176:A=PEEK(2034)
50 PRINTI;:FORJ=0TO7
60 IFAANDD(J)THENPRINT"1";:ELSEPRINT"0";
70 NEXT:PRINT:NEXT
80 GETA$:A$=A$+" ":CHAR,16,5,A$
90 PRINTCHR$(19):PRINT
100 LOOP
```

COMMENTO A TASTMAT.

.10: Cancella lo schermo e stampa una riga

.20: Crea il vettore D e lo riempie con le potenze del due, da 2^0 a 2^7. E' un "trucco" per velocizzare il programma, poiche' il calcolatore e' molto piu' rapido nell'accedere ad un elemento di un vettore che non nel calcolare un'elevamento a potenza.

.30: Inizializza un ciclo DO e un ciclo FOR. Il ciclo DO ... LOOP e' eseguito all'infinito, e FOR ... NEXT e' ripetuto 8 volte, e ogni volta analizza lo stato di ciascuna delle 8 righe di cui e' composta la tastiera.

.40: Usa la routine del SISTEMA OPERATIVO che pone un byte nel registro di uscita, quello collegato alle 8 righe, e che ritorna lo stato del registro di ingres-

so, quello cioe' collegato alle 8 colonne. Il contenuto del byte 2034 viene caricato nell'accumulatore della CPU al momento dell'istruzione SYS, e alla fine della routine il contenuto dell'accumulatore viene posto nel byte 2034. La routine del SISTEMA OPERATIVO, che inizia all'indirizzo 56176, pone nel registro di uscita (righe) il contenuto dell'accumulatore (il valore posto nel byte 2034 prima di eseguire la SYS, e che contiene tutti i bit a 1, tranne il bit di posizione I), e torna con l'accumulatore che contiene il valore del registro di ingresso (colonne). Pone inoltre questo valore, che si trova ancora nel byte 2034, nella variabile A.

.50: Si prepara a stampare nella riga che sta analizzando, e inizializza un ciclo da 0 a 7 che usa per controllare lo stato degli 8 bit del registro di ingresso.

.60: Se il bit che analizza e' a 1, stampa 1, altrimenti stampa 0, una volta per ciascun bit di ciascuna riga.

.70: Chiude i due cicli FOR, e stampa un RETURN alla fine del primo ciclo.

.80: Stampa il carattere premuto, nella posizione 16,5. Vengono stampati correttamente solo i tasti che rappresentano un carattere alfanumerico, e non quelli di controllo o di funzione.

.90: Posiziona il cursore all'inizio della seconda riga: CHR$(19) infatti e' il carattere HOME.

.100: Chiude incondizionatamente il ciclo DO ... LOOP che parte dalla linea 30. Avremmo potuto usare GOTO 30, ma DO ... LOOP e' un'istruzione piu' elegante e piu' rapida da eseguire.

La scansione della tastiera viene effettuata automaticamente dal SISTEMA OPERATIVO ogni cinquantesimo di secondo, ed e' per questo che abbiamo fatto uso della routine in linguaggio macchina, anziche' effettuare da BASIC la scrittura nel registro di riga e la lettura dal registro di colonna: tra l'istruzione di scrittura e quella di lettura il SISTEMA OPERATIVO avrebbe effettuato probabilmente qualche scansione della tastiera, cambiando il contenuto del registro di riga. Terminiamo l'argomento indicando che il registro di uscita (quello che si usa per selezionare una riga) risponde all'indirizzo 64816 (FD30H), e quello di ingresso, che si usa per vedere a quale colonna appartiene l'eventuale tasto premuto, risponde all'indirizzo 65288 (FF08H). La modalita' di funzionamento

di questo registro e' un po' particolare:
.occorre effettuare un'operazione di scrittura affinche' venga memorizzato lo stato delle linee esterne,
.effettuando dopo un ciclo di lettura e' possibile leggere lo stato delle linee esterne cosi' come era quando e' stato effettuato il ciclo di scrittura.
Questa complicazione e' stata necessaria per effettuare il collegamento con i joystick. Puoi verificare questa particolarita', se conosci gia' l'ASSEMBLER 6502, disassemblando la routine del SISTEMA OPERATIVO che noi richiamiamo nella linea 40 del programma TASTMAT, che parte dall'indirizzo 56176 (DB70H). Se non conosci ancora l'ASSEMBLER, puoi effettuare questa prova dopo aver letto il Capitolo 5.

4.6 I JOYSTICK

Al COMMODORE 16 si possono attaccare due joystick. Ogni joystick possiede 5 interruttori, 4 per le direzioni, alto, basso, sinistra, destra, e uno per il fuoco.

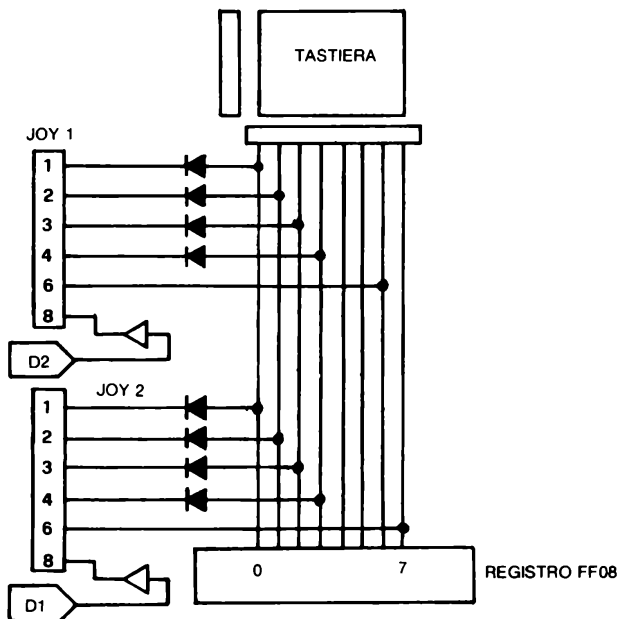



Figura 4.7a Collegamento con i Joystick

Nella Figura 4.7a e' riportato lo schema del collegamento con i joystick. Il piedino 8 dello spinotto del joystick 1 e' collegato al bit 2 del BUS DATI attraverso un BUFFER che serve solo per potenziare il segnale, prima di inviarlo all'esterno. Per leggere lo stato degli interruttori del joystick 1 e' sufficiente un ciclo di scrittura nel registro 65288 (FF08H), tale che il bit 2 del dato scritto sia a 0, e il bit 1 sia a 1. In questo modo se uno dei tasti del joystick 1 e' premuto, la linea collegata al piedino corrispondente a quel tasto risulta collegata al bit 2 del BUS DATI. Se si preme il bottone del fuoco il piedino 6 risulta collegato al bit 2. Poiche' siamo in un ciclo di scrittura, il bus dati contiene il valore del dato da scrivere, e in particolare il bit 2 contiene 0, come noi avevamo programmato. A questo punto il valore del registro di ingresso 65288 (FF08H) viene aggiornato in funzione dei suoi ingressi, e il bit 6 viene a trovarsi a 0, cosi' come e' a 0 il bit 2 del BUS DATI.

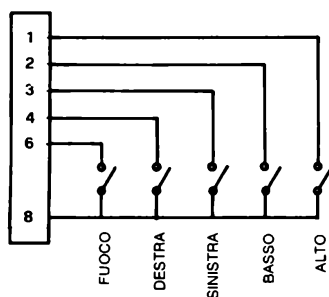



Figura 4.7b Schema elettrico dei Joystick

Nella Figura 4.7b e' riportato lo schema elettrico dei joystick. Le posizioni diagonali risultano dalla pressione contemporanea di due tasti, ad esempio "alto" piu' "sinistra" indicano che la leva del joystick e' posizionata in alto a sinistra. Nel Paragrafo 5.6 riportiamo una routine in linguaggio macchina che permette di leggere lo stato dei joystick.

4.7 IL REGISTRATORE A CASSETTE

Il registratore a cassette DATASSETTE 1531 si collega al COMMODORE 16 attraverso uno spinotto a 7 conness-

sioni. Ecco il significato di questi 7 punti di connessione:
.1: MASSA. La massa del calcolatore va collegata con quella del registratore attraverso questa connessione.
.2: +5V. Il registratore ha bisogno di un'alimentazione a 5 volt per i circuiti elettronici di pilotaggio della testina, che riceve da questo punto.
.3: MOTORE. Da questo punto il motore del registratore riceve la corrente necessaria per il suo funzionamento, 9 volt in corrente continua non stabilizzati. Questa alimentazione puo' venire interrotta dal calcolatore, per arrestare il motore.
.4: LETTURA CASSETTA. Su questa linea vengono trasmessi i dati da registratore a CPU durante la lettura di dati dal nastro.
.5: SCRITTURA CASSETTA. Durante la scrittura di dati su nastro magnetico i dati provenienti dalla CPU vengono trasmessi al registratore attraverso questa linea.
.6: SENSORE. Questa linea e' collegata ai tasti del registratore, perche' la CPU sia in grado di sapere se viene premuto qualche tasto.
.7: MASSA. Anche questa linea e' collegata a massa, come la linea 1.

Ora che abbiamo visto il significato delle linee che effettuano fisicamente il collegamento, vediamo come queste linee possono essere gestite.
Il programma TASTREG, mostra come si possa leggere lo stato dei tasti del registratore: il bit 2 del byte di indirizzo 64784 (FD10H) non e' un bit di memoria, ma un bit di ingresso, il cui stato rispecchia lo stato dei tasti del registratore. Se tale bit e' a livello logico 1 nessuno dei 3 tasti PLAY, REWIND, FFWD e' premuto, altrimenti almeno uno di essi e' premuto.

```
0 REM TASTREG
10 DO
20 PRINT"PREMI UN TASTO SUL REGISTRATORE"
30 DO:LOOP WHILE PEEK(64784)AND4
40 PRINT"PREMI STOP SUL REGISTRATORE"
50 DO:LOOP UNTIL PEEK(64784)AND4
60 LOOP
```

COMMENTO A TASTREG
.10: Inizializza un ciclo DO ... LOOP infinito, per ripetere all'infinito il programma.
.20: Stampa un messaggio sul video, ad indicare che ora nessun tasto e' premuto.

.30: Attende che venga premuto un tasto sul registratore.
.40: Stampa un messaggio sul video, ad indicare che ora e' premuto almeno un tasto.
50: Attende che vengano rilasciati tutti i tasti del registratore.
60: Chiude il ciclo iniziato alla linea 10.

Abbiamo visto come scoprire se un tasto del registratore e' premuto oppure no. Il programma che segue, MOTREG, mostra come si possa arrestare o far partire il motore.

```
0 REM MOTREG
10 SCNCLR:PRINT"PREMI PLAY SUL REGISTRATORE"
20 PRINT:PRINT"A = MOTORE ACCESO"
30 PRINT"S = MOTORE SPENTO"
40 PRINT"F = FINE"
50 GOSUB120
60 GETKEYA$
70 IFA$="A"THENGOSUB120
80 IFA$="S"THENGOSUB150
90 IFA$<>"F"THEN60
100 GOSUB120
110 END
120 POKE0,PEEK(1)OR8
130 CHAR,0,6,"MOTORE ACCESO"
140 RETURN
150 POKE0,PEEK(1)AND247
160 CHAR,0,6,"MOTORE SPENTO"
170 RETURN
```

COMMENTO A MOTREG
.10/40: Stampa la videata con il menu'.
.50: All'inizio il motore viene acceso.
.60: Attende la pressione di un tasto sulla tastiera.
.70: Se il tasto premuto e' A accende il motore.
.80: Se il tasto premuto e' S spegne il motore.
.90: Se il tasto premuto non e' F torna alla linea 60.
.100: Prima di finire, accende il motore.
.110: Termina l'esecuzione del programma.
.120/140: Subroutine che accende il motore ponendo a 1 il bit 3 del byte 1 e stampa il messaggio "MOTORE ACCESO". Il motore e' collegato al bit 3 della porta di I/O inserita nella CPU 7501. Tale porta di I/O risponde all'indirizzo 1.
150/170: Subroutine che spegne il motore azzerando il bit 3 del byte 1 e stampa il messaggio "MOTORE SPENTO".
Dopo l'esecuzione del programma e' meglio eseguire un RESET del calcolatore.

Con questi due programmi abbiamo visto come si gestiscono dei dettagli sul registratore, i tasti e il motore, e non vediamo come effettivamente sono gestiti i dati. Il software di gestione della linea seriale del registratore e' troppo complesso per poter produrre riguardo ad esso degli esempi semplici: possiamo tuttavia illustrare i tipi di segnali che viaggiano sulle linee dati seriali da e verso il registratore. Le due linee LETTURA CASSETTA e SCRITTURA CASSETTA sono due linee digitali normali, e su di esse viaggiano segnali che possono assumere il valore 0 o 1: chi e' esperto di registrazioni magnetiche forse trova un po' strano questo fatto, dato che ci si sarebbe potuti aspettare un segnale sinusoidale, in cui una frequenza corrispondesse al livello logico 1 e un'altra frequenza al livello logico 0, con una modulazione in scrittura ed una demodulazione in lettura. Questo tipo di trasmissione, usato in diversi calcolatori, non viene usato nei calcolatori COMMODORE. Una semplice elettronica di pilotaggio nel registratore invia gli stessi livelli logici che riceve in ingresso alla testina, polarizzando in una direzione il nastro quando il livello logico e' alto, e nella direzione opposta quando il livello e' basso.

Nella Figura 4.8a e' riportata una semplificazione dello schema elettrico di pilotaggio della testina all'interno del registratore, durante la fase di registrazione, e nella Figura 4.8b sono illustrati i livelli di tensione applicati alla testina e la magnetizzazione corrispondente del nastro. Il tipo di

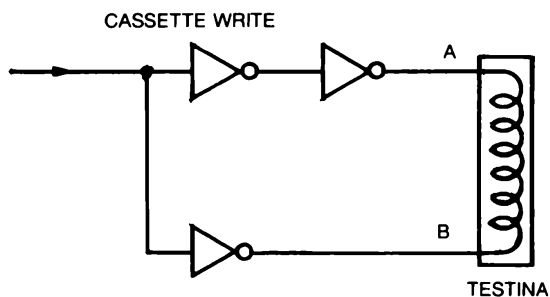



Figura 4.8a Collegamento della testina all'interno del registratore durante la fase RECORD

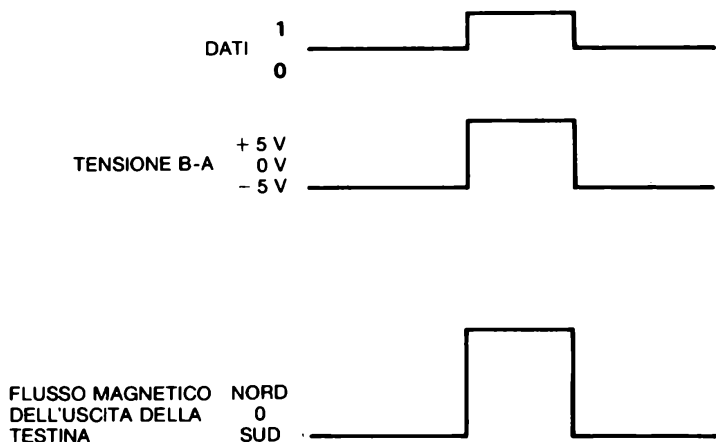



Figura 4.8b Livelli di tensione e magnetizzazione della testina del registratore

collegamento adottato permette di avere all'uscita della testina (e quindi su nastro), sempre una polarizzazione magnetica, verso nord o verso sud, sia in corrispondenza del livello 0 che in corrispondenza del livello 1. Se si fosse collegato il punto A o il punto B (vedi schema in Figura 4.8a) alla massa, in corrispondenza di uno dei livelli logici 0 o 1 ci sarebbe stata assenza di polarizzazione (anziche' polarizzazione in senso opposto), con conseguente instabilita' in fase di lettura dati. Il livello di registrazione risulta percio' essere sempre quello di saturazione del nastro, e per questa ragione conviene

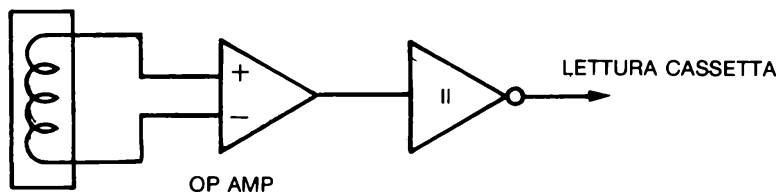



Figura 4.9 Collegamento della testina in fase PLAY

usare nastri ad alto livello di saturazione (HI ENERGY) per aumentare l'affidabilita' delle registrazioni.

Nella Figura 4.9 riportiamo un'esemplificazione di come e' collegata la testina in fase di lettura: un amplificatore a guadagno elevato amplifica il segnale della testina e lo manda all'ingresso di un INVERTER che pilota la linea fino al registratore. La parte iniziale, l'ingresso dell'amplificatore, e' molto sensibile ai disturbi, per cui e' raccomandabile, soprattutto in fase di PLAY, tenere il registratore distante da fonti di disturbo, e in particolare distanti dallo schermo video, soprattutto se grande e a colori. La linea SCRITTURA CASSETTA e' collegata al bit 1 del byte 1, e la linea LETTURA CASSETTA e' collegata al bit 4 della porta 1. Il programma VEDEBIT stampa su video lo stato della linea LETTURA CASSETTA. Quando il registratore e' fermo o sulla cassetta non e' inciso segnale, il numero stampato su video non cambia, mentre durante l'ascolto di un pezzo di nastro registrato, vedi che a volte viene stampato 0, a volte 16 (2^4), a indicare che il bit 4 e' ora nello stato 0, ora nello stato 1. Non e' possibile in BASIC ricavare informazioni dal nastro, a causa della lentezza e della mancanza di temporizzazioni precise tipici del linguaggio.

```
0 REM VEDEBIT
10 PRINT"PREMI PLAY SUL REGISTRATORE'
20 DO:PRINTPEEK(1)AND16,:LOOP
```

COMMENTO A VEDIBIT
 .10: Stampa il messaggio
 .20: Stampa all'infinito il risultato della operazione logica AND tra 16 e il contenuto del byte 1, per considerarne solo il bit 4 (2^4=16). Nel Paragrafo 5.4.1 parliamo dettagliatamente di questa e di altre operazioni tra bit.

4.8 IL VIDEO

Il video e' diventato elemento essenziale nei calcolatori, poiche' da' una caratteristica di interattivita' assai gradevole ed utile sia nello scrivere

che nell'eseguire i programmi. Vediamo come funziona uno schermo video: si tratta di uno schermo ricoperto di elementi (FOSFORI) che quando sono colpiti da elettroni emettono luce. Dietro questo schermo vi e' un "cannone" che emette un fascio di elettroni di intensita' regolabile verso il centro dello schermo. Poiche' gli elettroni sono carichi elettricamente, possono essere deviati da un campo elettrico. Per deviare gli elettroni vi sono delle placche sui 4 lati dello schermo: sopra, sotto, a destra e a sinistra. A seconda del potenziale applicato alle placche orizzontali si puo' orientare il fascio di elettroni verticalmente, mentre variando il potenziale applicato alle placche verticali lo si puo' orientare orizzontalmente. Per ottenere un'immagine sul video si e' diviso lo schermo in 312 righe orizzontali, e si fa passare il fascio di elettroni da sinistra verso destra dalla prima fino alla trecentododicesima riga in un cinquantesimo di secondo. Un opportuno segnale regola poi l'intensita' del fascio che esce dal cannone, e cio' permette di avere punti diversamente luminosi sopra la stessa riga.

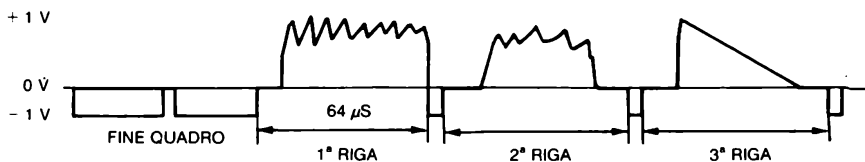



Figura 4.10 Segnale video per pilotare un monitor

Nella Figura 4.10 e' illustrato il formato del segnale video adatto a pilotare un normale monitor: la scansione di una riga dura 64 microsecondi. Il cannone "spara" elettroni in misura proporzionale al livello del segnale, istante per istante. Per disegnare una riga illuminata il segnale deve avere un livello di tensione alto per tutto il tempo in cui la riga deve essere illuminata, e il livello basso del segnale corrisponde al cannone che emette pochi elettroni, in modo da non far illuminare i fosfori che colpisce. I picchi di segnale negativi rappresentano i

SINCRONISMI, e informano il monitor su quando andare a nuova riga, e quando andare a nuovo quadro. Il sincronismo di riga ha la frequenza di 15625 hertz, cioe' il periodo dura 64 microsecondi, e il sincronismo di quadro ha la frequenza di 50 hertz, cosicche' il fascio elettronico ritorna sullo stesso punto ogni 20 millisecondi.
Nel COMMODORE 16 c'e' un circuito integrato assai complesso che genera il segnale video. Tale integrato e' stato progettato appositamente dalla COMMODORE, porta la sigla 7360 ed e' stato battezzato TED, come TExt Display, perche' si occupa principalmente della visualizzazione del testo. Vediamo quali sono i compiti di TED e come li esegue: sappiamo che una pagina video di quelle che TED deve saper visualizzare e' formata da 40 X 25 caratteri. Poiche' ogni carattere occupa una matrice di 8 X 8 puntini (PIXEL) occorrono 320 punti su una riga. TED ha a disposizione circa 45 microsecondi (una parte della riga e' occupata dal bordo) per visualizzare una riga di testo, cioe' 140 nanosecondi (miliardesimi di secondo) per pixel. Come forse hai notato TED e' estremamente rapido. Ma vediamo come TED viene a conoscenza dei dati che deve visualizzare: tutte le informazioni di cui ha bisogno le puo' trovare nella memoria del calcolatore, la stessa che e' a disposizione della CPU. Per mostrare una riga TED ha bisogno di una quantita' rilevante di dati, in poco tempo:

. Il codice (D/CODE, vedi Appendice D) del carattere da visualizzare, 40 volte ogni otto righe.
. Il colore e la luminosita' del carattere, 40 volte ogni 8 righe.
. L'immagine del carattere da visualizzare, 40 volte ogni riga.

Deduciamo che per visualizzare la prima di otto righe TED deve fare accesso a memoria ben 120 volte, mentre per le altre 7 righe bastano 40 accessi ciascuna, che vengono effettuati durante la fase bassa di Φ0 (il D/CODE del carattere e gli attributi sono gli stessi per 8 righe). Quando accede al codice e agli attributi TED disabilita la CPU; cio' comporta un rallentamento nell'esecuzione di tutti i programmi. Si puo' programmare TED in maniera che non disturbi la CPU, azzerando il bit 4 del registro FF06H (65286 decimale), ma in questo caso non puo' visualizzare lo schermo, che appare tutto del colore del bordo. Ti ricordiamo che TED accede a memoria senza disabilitare la CPU durante la fase bassa di Φ0 (vedi Figura 4.5), ed

utilizza questo tempo per leggere le immagini dei caratteri da visualizzare.

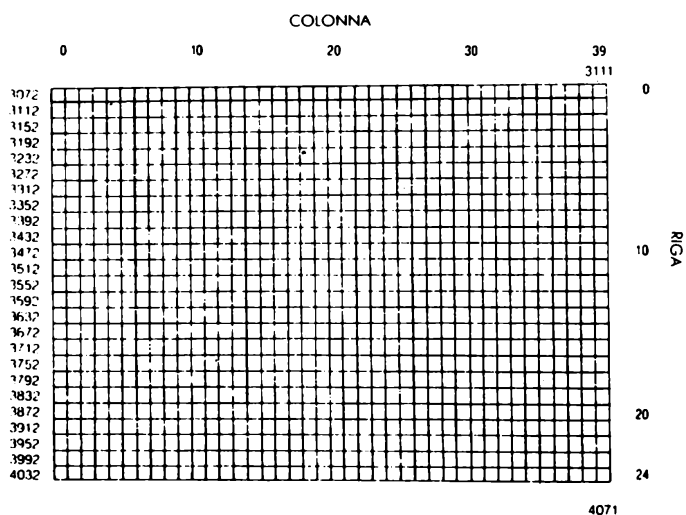



Figura 4.11a Mappa della memoria dei caratteri

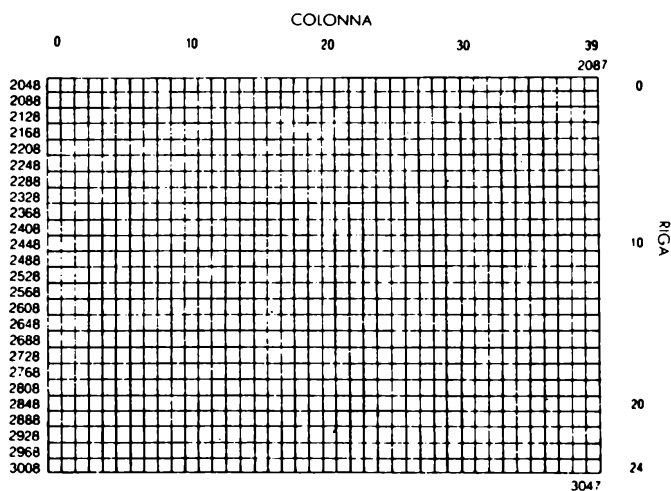



Figura 4.11b Mappa della memoria del colore e della luminosita' dei caratteri

Nella Figura 4.11a sono elencate le locazioni di memoria che controllano la disposizione dei caratteri sullo schermo e nella Figura 4.11b le locazioni che controllano gli attributi, cioe' il colore, il livello di luminosita' e il lampeggiamento dei caratteri. Sono queste, oltre all'immagine dei caratteri, residente in ROM, le zone di memoria da cui il TED attinge le informazioni necessarie per effettuare la visualizzazione dello schermo nel modo testo. La memoria degli attributi contiene i dati relativi alla luminosita' (i bit 6-5-4), al colore (i bit 3-2-1-0) e al lampeggiamento (il bit 7) di ciascun carattere. Gli indirizzi segnati nella figura sono quelli validi all'accensione del calcolatore, e vedrai nel Capitolo 6 come si possono modificare. La programmazione del TED e' assai laboriosa, soprattutto a causa del gran numero di funzioni che puo' eseguire, e per questo affrontiamo l'argomento in diversi punti, particolarmente nel Capitolo 6 dedicato alla grafica.
TED ha un numero di funzioni assai elevato all'interno del COMMODORE 16: genera il CLOCK per la CPU, genera il segnale video, genera il segnale audio, rinfresca le memorie dinamiche e regola le temporizzazioni necessarie per la gestione di tali memorie, riceve i dati che provengono dalla tastiera e quelli che provengono dai joystick, genera gli INTERRUPT per la CPU, emette i segnali di abilitazione per le ROM, e infine seleziona il banco ROM o RAM negli indirizzi 8000H-FFFFH. Come vedi e' riduttivo pensare al TED solo come interfaccia per il video: il TED e' un integrato che effettua gran parte delle funzioni di I/O e di interfacciamento con la memoria.
Il segnale video esce dal calcolatore dalla presa marcata VIDEO, presente sul retro della tastiera. Il piedino 1 emette un segnale che contiene LUMINANZA e SINCRONISMI, ma non contiene le informazioni relative al colore, che escono invece dal piedino 6 (CROMINANZA). Il piedino 4 contiene un segnale VIDEO COMPOSTO, che comprende LUMINANZA, CROMINANZA e SINCRONISMI. Al piedino 1 si puo' attaccare un monitor monocromatico; il monitor a colori della COMMODORE modello 1701/1702 fa uso dei segnali separati LUMINANZA+SINCRONISMI e CROMINANZA, che preleva dai piedini 1 e 4 dello spinotto, mentre i monitor di altre marche generalmente fanno uso del segnale VIDEO COMPOSTO presente al piedino 4. La qualita' del segnale VIDEO COMPOSTO e' inferiore a quella dei segnali separati LUMINANZA e CROMINANZA, ed e' anche per que-

sto che la qualita' dell'immagine su monitor COMMODORE 1701 e' migliore di quella di quasi tutti gli altri monitor della stessa fascia di prezzo. Il segnale per il video esce anche da un altro spinotto, quello marcato RF, modulato con radiofrequenza. Tale segnale e' adatto all'uso con un normale televisore, ma la qualita' dell'immagine e' generalmente peggiore di quella del monitor. Il televisore va sintonizzato sul canale 36, in banda IV (UHF), e talvolta puo' essere necessario ritoccare la sintonia del TV a causa di inevitabili slittamenti della frequenza che dipendono dalla variazione di temperatura dei componenti del modulatore. La Figura 4.12 riporta lo schema di collegamento dello spinotto VIDEO.

PIEDINO	FUNZIONE
1	LUMINANZA + SINCRONISMI
2	MASSA
3	USCITA AUDIO
4	VIDEO COMPOSTO
5	AUDIO IN
6	CROMINANZA
7	NON COLLEGATO
8	+ 5 VOLT

Figura 4.12 Connessioni della presa AUDIO/VIDEO

4.9 IL SUONO

Il suono nel COMMODORE 16 viene generato da due oscillatori che si trovano nel TED. Per ognuno di questi vi e' un registro a 10 bit che indica la frequenza di

oscillazione; vi e' poi un registro comune, di quattro bit, che indica il volume del suono emesso: il livello 0 corrisponde a silenzio, e il livello 8 al massimo (i livelli da 8 a 15 sono equivalenti). Tre bit servono ad azionare le due voci, e uno di questi indica se la voce 2 emette un segnale a onda casuale (rumore bianco).

FREQ.: VOCE 1	9 8 BIT 1 E 0 $ FF12	7 6 5 4 3 2 1 0 F F 0 F
FREQ.: VOCE 2	9 8 BIT 1 E 0 $ FF.10	7 6 5 4 3 2 1 0 F F 0 F

ABILITAZIONE: BIT 4 $ FF11 (0 = OFF)
VOCE 1

ABILITAZIONE: BIT 5 $ FF11 (0 = OFF)
VOCE 2

ABILITAZIONE:
RUMORE BIANCO: BIT 6 $ FF11 (0 = OFF)
PER VOCE 2

DISABILITAZIONE: BIT 7 $ FF11 (1 = DISABILITATO)
DEL SUONO

VOLUME: BIT 03 $ FF11

Figura 4.13 Registri per la gestione del suono

Nella Figura 4.13 sono riportati gli indirizzi e le posizioni di tutti i bit che riguardano il suono. Per generare rumore bianco e' sufficiente selezionare il bit del rumore bianco, e non e' necessario selezionare anche il bit di abilitazione della voce 2, dopo aver impostato la frequenza.
Il programma SUONOCONPOKE e' un esempio di come vanno gestiti i registri di TED per generare suoni.

```
0 REM SUONOCONPOKE
10 T=255*256
20 POKET+17,8+32
30 FORF=1TO4*256-1STEP10
40 POKET+15,FAND255:POKET+16,F/256:NEXT
50 POKET+15,200:POKET+16,0
60 FORV=8TO0STEP-1:POKET+17,V+32
70 FORR=1TO200:NEXT:NEXT
```

Il suono esce dallo spinotto marcato VIDEO presente sul retro della tastiera, ed e' collegato al piedino 3. Per chi usa il televisore il segnale del suono e' modulato, e appare nell'altoparlante del TV come quello di un normale canale televisivo. Nella Figura 4.12 e' riportato lo schema dello spinotto marcato VIDEO. Il piedino 5 dello spinotto, AUDIO IN, permette di inserire un segnale audio in ingresso: il segnale audio in uscita AUDIO OUT comprende il mixaggio di AUDIO IN e del suono generato dal TED.

4.10 LA PORTA SERIALE

Se per i calcolatori stai acquistando quella sorta di passione che ha conquistato ed anima noi che scriviamo, ti troverai, prima o poi, nella necessita' di espandere il tuo calcolatore con il DISK DRIVE e con la STAMPANTE: potrai usare il tuo sistema calcolatore per gestire archivi elettronici, per scrivere lettere e libri (questo libro e' stato scritto su COMMODORE 64, con il programma di elaborazione testi EASY SCRIPT, e stampato con una stampante a margherita: EASY SCRIPT sara' presto disponibile anche per COMMODORE 16), e per automatizzare quelle operazioni che riterrai possibile e utile automatizzare. Il tuo COMMODORE 16 potra' ancora esserti utile, poiche' le

sue doti di espandibilita' e i programmi che gradualmente vengono presentati sul mercato ti permetteranno di usare questo calcolatore anche per usi "professionali". La porta seriale permette di collegare FLOPPY DISK DRIVE e STAMPANTE, oltre ad eventuali altre apparecchiature (PLOTTER, ecc) che ora o in seguito potranno essere costruite.

PIEDINO	FUNZIONE
1	NC
3	SERIAL ATN OUT
6	$\overline{\text{RESET}}$ OUT
2	MASSA

VISTA FRONTALE
DELLA PRESA (FEMMINA)

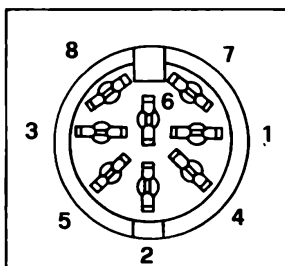



Figura 4.14 Schema della porta di I/O seriale
(SERIAL BUS)

Vediamo nella Figura 4.14 lo schema di collegamento della porta di I/O seriale (SERIAL BUS). I nomi dei segnali sono in inglese, poiche' nella terminologia corrente se ne fa uso in lingua inglese.
Il collegamento tra calcolatore e periferiche su SERIAL BUS e' abbastanza complesso, tuttavia possiamo illustrare il principio logico di funzionamento, e il significato dei segnali che realizzano il collegamento. Il SERIAL BUS e' stato progettato in maniera da permettere il collegamento con un numero e un tipo variabile di periferiche. Se osservi attentamente la Figura 4.14 ti accorgi che i fili su cui effettivamente viaggiano segnali utili per lo scambio delle informazioni sono 3, piu' la massa. Attraverso questi 4 fili e' possibile comunicare con un numero di periferiche che puo' arrivare anche a 7 (2 STAMPANTI, numero 4 e 5, un PLOTTER, numero 6, quattro DISK DRIVE, numeri da 8 a 11. Appare evidente come vi debba essere un accorgimento per comunicare con 7 diverse unita' attraverso 4 soli fili. Poniamo l'attenzione sul SERIAL BUS, cioe' su questi 4 fili a cui risultano collegati, per fare un esempio: CALCOLATORE, DISK DRIVE e STAMPANTE.
E' necessario che i dati siano in grado di viaggiare nelle seguenti direzioni:
.da calcolatore a stampante,
.da calcolatore a drive,
.da drive a calcolatore.
Non e' previsto un collegamento in cui il calcolatore sia escluso, cioe' da drive a stampante. E' stato stabilito che sul SERIAL BUS possono essere collegati tre tipi di dispositivi:
- CONTROLLER, cioe' il dispositivo che controlla il flusso dei dati; chi assume questa funzione e' sempre il calcolatore,

- TALKER, cioe' il dispositivo che invia i dati,
- LISTENER, cioe' il dispositivo che riceve i dati.

Inoltre un dispositivo puo' essere in uno stato in cui ignora i dati che viaggiano sul bus.
Per effettuare la trasmissione dei dati il controller invia dei messaggi agli apparecchi collegati, e attende la loro risposta. Ogni apparecchio ha un numero che lo identifica: la STAMPANTE, ad esempio, normalmente ha il numero 4. Per comunicare con la stampante il calcolatore invia all'unita' 4 l'ordine di ascoltare (diventare LISTENER). Se l'unita' 4 risponde entro un ventesimo di secondo, il calcolatore invia i dati e la

stampante li stampa, altrimenti viene rilevato un errore di mancanza del dispositivo (DEVICE NOT PRESENT). Alla fine di ogni dato la stampante invia una risposta, per informare il calcolatore che il dato e' stato accettato, per evitare che si perdano dei dati a causa del fatto che la stampante e' piu' lenta a stampare di quanto lo sia il calcolatore ad inviare i caratteri. Alla fine della stampa il calcolatore comanda alla stampante di uscire dallo stato di LISTENER, e di ignorare gli eventuali caratteri futuri (potranno anche non essere indirizzati a lei).
Il software per il collegamento con il DRIVE e' un po' piu' complesso, perche' il DRIVE, a differenza della stampante, e' in grado sia di ricevere, sia di trasmettere dati: il modo di ricezione dei dati e' uguale a quello della stampante, mentre la trasmissione da DRIVE a CALCOLATORE avviene nel modo seguente: il calcolatore, che ha bisogno di informazioni dal drive, trasmette al drive i dati di cui ha bisogno (ad esempio, il nome del file che vuole caricare in memoria), dopodiche' comanda al drive di diventare TALKER (colui che parla). Il drive invia tutti i caratteri del file (il calcolatore intanto ascolta), e alla fine trasmette una sequenza di FINE TRASMISSIONE. Il calcolatore comanda percio' al drive di non trasmettere e non ascoltare piu' i dati, in modo da liberare la linea seriale per i collegamenti anche con le altre periferiche.
Come forse hai gia' notato, il calcolatore e' in grado di comandare alle unita' presenti su linea seriale di ascoltare, di parlare, di ignorare. Tutte queste informazioni speciali possono partire solo dal calcolatore, e sono segnali trasmessi sotto ATTENTION. Il SERIAL ATN OUT e' un segnale che esce dal calcolatore, e significa che i dati che viaggiano sulle linee CLOCK e DATA sono da interpretarsi come comandi. In questo modo il calcolatore trasmette due byte da otto bit: il primo contiene il numero di periferica a cui e' inviato il comando (4 bit meno significativi) e il tipo di comando da eseguire (LISTEN, TALK, UNLISTEN, UNTALK, 4 bit piu' significativi); il secondo byte contiene un numero chiamato INDIRIZZO SECONDARIO che la periferica riceve, ed ha un significato che dipende dalla periferica a cui e' diretto.
Quando e' attivo il segnale ATTENTION nessuna periferica presenta dati sulle linee SERIAL CLOCK e SERIAL DATA, in modo da permettere al calcolatore di eseguire la sequenza di ATTENTION, e tutti i dispositivi

ascoltano, pronti a eseguire i comandi inviati. Come puoi osservare nella Figura 4.15, tutte le linee sono di tipo OPEN COLLECTOR, cioe' l'uscita di ogni dispositivo collegato e' in grado di porre la linea BASSA se la linea e' alta, ma nessuna uscita e' in grado di riportare la linea ALTA, se qualche altra unita' la tiene bassa.

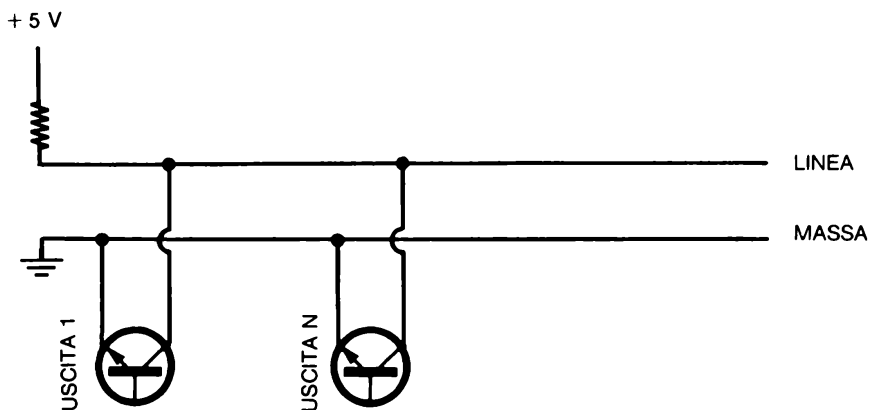



Figura 4.15 Schema elettrico del SERIAL BUS

E' chiaro che la linea e' libera quando nessun dispositivo la pone bassa, cioe' quando tutte le uscite sono alte. Se non conosci il funzionamento di un transistor, puoi immaginare i transistor disegnati in corrispondenza di ciascuna uscita come degli interruttori: se l'interruttore e' chiuso la linea si trova collegata alla massa, e il livello e' BASSO; se l'interruttore e' aperto lo stato della linea puo' essere basso o alto, e dipende dagli altri interruttori: se tutti gli interruttori sono aperti, allora la linea assume il livello alto, a causa del resistore collegato al positivo di alimentazione.

4.11 L'ORGANIZZAZIONE DELLA MEMORIA

Nella Figura 4.16 e' illustrata l'organizzazione della memoria nel COMMODORE 16. Abbiamo visto nei paragrafi precedenti che questo calcolatore e' dotato di 16K BYTE di memoria RAM. Il calcolatore e' stato progettato in modo da poterne pero' gestire 64K BYTE.

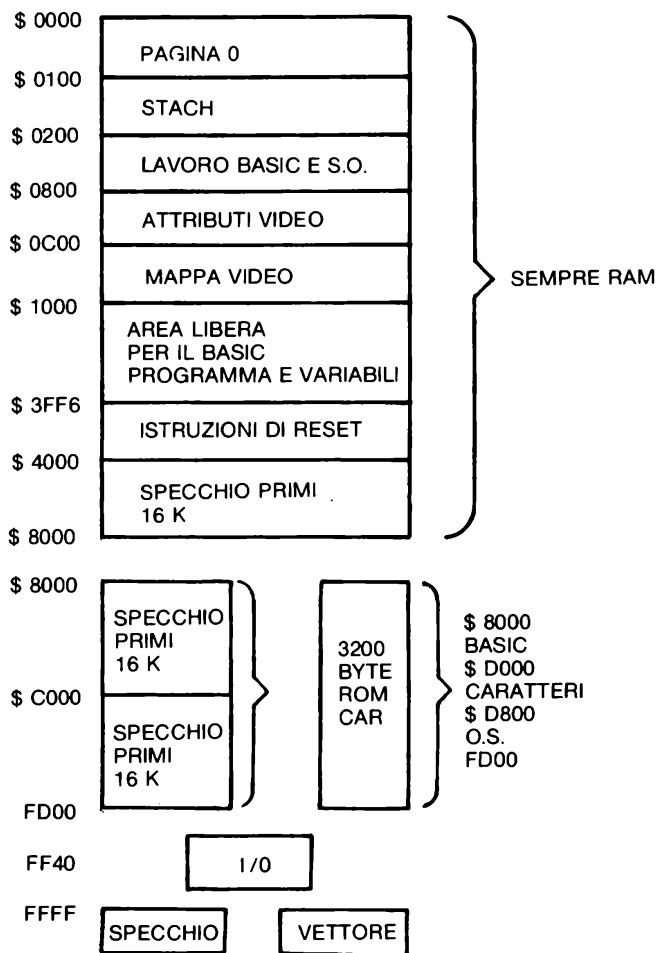



Figura 4.16 Organizzazione della memoria

Se osservi bene la Figura 4.16 puoi notare che i primi 16K di memoria RAM si ripetono 4 volte, cioe' i secondi, i terzi e i quarti 16K sono uguali ai primi 16K, perche' ogni byte di memoria risponde a 4 indirizzi. Il byte di indirizzo 3072, ad esempio, risponde anche agli indirizzi 3072+16384, 3072+32768, 3072+49152. Il programma RAM & ROM, che segue, mostra come a 4 indirizzi diversi corrisponde lo stesso byte.

```
0 REM RAM&ROM
10 FORJ=4096 TO 4099
20 FORI=0 TO 3
30 PRINT"LA CELLA" I*16384+J;
40 PRINT"CONTIENE";PEEK(I*16384+J)
50 NEXT I
60 PRINT:NEXT J
```

COMMENTO A RAM&ROM

.10 Stampa i contenuti dei byte che vanno da 4096 a 4099, ma tutti gli altri byte di indirizzo inferiore a 16384 possono essere analizzati con lo stesso risultato

.20-50 Ripete 4 volte questo ciclo, per stampare il contenuto dei 4 blocchi di indirizzi a cui rispondono i 16K di RAM del COMMODORE 16

.60 Stampa un RETURN per separare i risultati e chiude il ciclo di J.

Considerando il fatto che la CPU dispone di soli 16 fili per indirizzare tutte le memorie collegate (ROM, RAM, I/O), puo' cioe' indirizzare al massimo 64K di memoria RAM, e i 64K risultano occupati da 4 blocchi da 16K di RAM, e' lecito chiedersi dove siano i 32K di ROM. e dove siano i registri di I/O. Sempre guardando la Figura 4.16 puoi osservare come gli indirizzi di I/O siano comuni a tutte le mappe, partendo da FD00H fino a FF40H. La ROM puo' alternarsi alla RAM negli indirizzi da 8000H a FFFFH (escluso l'I/O). Per abilitare la ROM a rispondere agli indirizzi suddetti si deve effettuare un'operazione di scrittura nel registro del TED che risponde all'indirizzo FF3EH, mentre per abilitare la RAM basta effettuare la scrittura nel registro di indirizzo FF3FH.

CAPITOLO 5

LA PROGRAMMAZIONE IN ASSEMBLER E IN LINGUAGGIO MACCHINA

5.1 INTRODUZIONE

Nel Capitolo 4 abbiamo esposto le caratteristiche HARDWARE del COMMODORE 16 anche per metterti in condizione di utilizzare i dispositivi che lo compongono programmando in ASSEMBLER. Uno degli ostacoli piu' grossi nell'apprendimento della programmazione a BASSO LIVELLO e' quello di far comunicare il calcolatore con l'esterno. L'ASSEMBLER e' a basso livello perche' vicino alla macchina, lo e' ancora di piu' il LINGUAGGIO MACCHINA; alto livello e' quello dei linguaggi vicini all'uomo, come BASIC, PASCAL, FORTRAN, ecc.. In questo capitolo vediamo come e' fatta la CPU 7501, quali istruzioni e' in grado di eseguire, come e' fatto e come viene eseguito il programma.
Il programma che la CPU esegue e' contenuto in memoria come una sequenza di byte. La CPU preleva dalla memoria il byte, che rappresenta l'istruzione da eseguire, e gli eventuali operandi indicati nei byte seguenti; esegue l'istruzione e passa a prelevare l'istruzione successiva. Ogni istruzione e' quindi letta dalla memoria come un normale byte. Programmare in linguaggio macchina vuol dire porre nei byte di memoria i codici delle istruzioni che si vogliono eseguire, e gli operandi. Come puoi immaginare, e' assai laborioso scrivere un programma sotto forma di numeri (in binario), e per questo non e' quasi mai usata questa tecnica.
Il linguaggio macchina lavora su dati numerici contenuti in un byte, quindi con valore da 0 a 255. Per poter trattare numeri piu' grandi o con segno si devono

usare sottoprogrammi adatti. Lo stesso discorso vale per trattare numeri con cifre dopo il punto decimale (floating-point) e dati alfanumerici.
Le istruzioni che la CPU riconosce hanno ricevuto un nome mnemonico che si preferisce usare per scrivere il programma. Esistono dei programmi (alcuni chiamati ASSEMBLATORI), che traducono in codice oggetto (combinazione di zeri e uni da porre in memoria) il programma redatto da noi in forma mnemonica (che viene chiamato programma sorgente). Il comando MONITOR del BASIC 3.5 pone a nostra disposizione proprio un programma traduttore, anche se non si tratta di un vero assemblatore. Parliamo diffusamente del MONITOR nel Paragrafo 5.6.
Anche se, grazie al MONITOR, non dobbiamo tradurre in numeri i codici delle istruzioni, abbiamo a che fare con i byte della memoria per gli operandi delle istruzioni e per gli indirizzi. Il sistema di numerazione decimale non consente una conversione immediata in binario. D'altra parte il sistema binario richiede troppe cifre per indicare un byte: ad esempio il numero 221 decimale corrisponde al numero binario 11011101. Il sistema di numerazione che e' stato scelto per indicare gli operandi delle istruzioni e' quello a base 16, o ESADECIMALE. In tale sistema vi sono 16 cifre: 0 1 2 3 4 5 6 7 8 9 A B C D E F. Il vantag-

ESADECIMALE	BINARI	DECIMALE
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Figura 5.1 Esadecimale, binario e decimale

gio dell'esadecimale rispetto al decimale e' che un numero esadecimale si puo' tradurre in binario piu' in fretta, perche' ogni cifra rappresenta 4 bit; rispetto al binario il vantaggio e' che i numeri sono piu' compatti. In questo libro e nei libri di ASSEMBLER 6502 i numeri esadecimali sono preceduti dal segno dollaro: il numero $23 si rappresenta in binario 00100011.
Nella Figura 5.1 trovi la corrispondenza dei valori delle cifre esadecimali nei sistemi binario e decimale. Come nel sistema decimale dopo il numero 9 si passa a 10 ponendo 1 nella posizione piu' a sinistra, cosi' nell'esadecimale dopo $F si passa a $10, che vuol dire 16 in decimale. La Figura 5.2 mostra ancora un esempio di conversione: lo stesso numero 101 viene analizzato prima come binario, poi come esadecimale e poi come decimale. A titolo informativo aggiungiamo che nel mondo dei calcolatori si fa spesso uso anche del sistema a base 8, detto OTTALE, dove le cifre vanno da 0 a 7. Noi non faremo mai riferimento a tale sistema in questo libro, e per questo non lo approfondiamo.

Figura 5.2 Analisi del numero 101

Accenniamo, da ultimo, al sistema BCD (Bynary Coded Decimal), cioe' decimale a codifica binaria: e' una via di mezzo tra esadecimale e decimale, perche' ogni cifra BCD rappresenta 4 bit (come l'esadecimale) e le cifre vanno da 0 a 9 (come il decimale). Il difetto di questo sistema di rappresentazione e' un notevole

spreco di bit: un byte puo' infatti contenere solo 100 combinazioni ammesse (da 0 a 99), invece di 255 possibili in binario. Un altro svantaggio risiede nell'elaborazione: consideriamo come esempio una qualsiasi operazione di somma:

```
NUMERO   RAPPRESENTAZIONE BCD
  08 +         00001000 +
  12 =         00010010 =
------         ----------
  20           00100000 invece di 00011010
                        come deve essere.
```

Osserviamo subito che l'unita' di calcolo all'interno della CPU deve lavorare diversamente se gli operandi e il risultato sono in BCD da come opera con gli operandi in binario. Nella 7501 e' possibile selezionare il modo BCD con un'istruzione, e tornare al modo binario con un'altra istruzione, e questa possibilita' aiuta notevolmente il programmatore ASSEMBLER nel gestire i dati decimali.

5.2 ORGANIZZAZIONE DELLA CPU 7501

La 7501 e' una CPU che deriva dalla famiglia 6500, una serie di unita' centrali di elaborazione sviluppata negli anni '70 negli USA dalla ROCKWELL INTERNATIONAL CORPORATION. La "filosofia" dei progettisti e' stata quella di creare una CPU dotata di un numero di istruzioni relativamente limitato, un numero di registri interni anch'esso limitato, ma di potenti e veloci modi di indirizzamento. Per limitare il numero di istruzioni non si sono poste istruzioni specifiche per gestire l'I/O: la CPU gestisce solo la memoria, e l'I/O deve essere collegato alla CPU come la memoria. Il limitato numero di registri e' compensato dalla gestione rapida dei dati in PAGINA ZERO: le celle di memoria che partono dall'indirizzo 2 fino a $00FF sono da considerarsi come un'estensione dei registri interni della CPU, poiche' si possono indirizzare piu' rapidamente e permettono indirizzamenti assai potenti, come vedremo nel prossimo paragrafo.
Osserviamo la Figura 5.3 per vedere come funziona internamente la CPU. Partiamo dai registri: la 7501

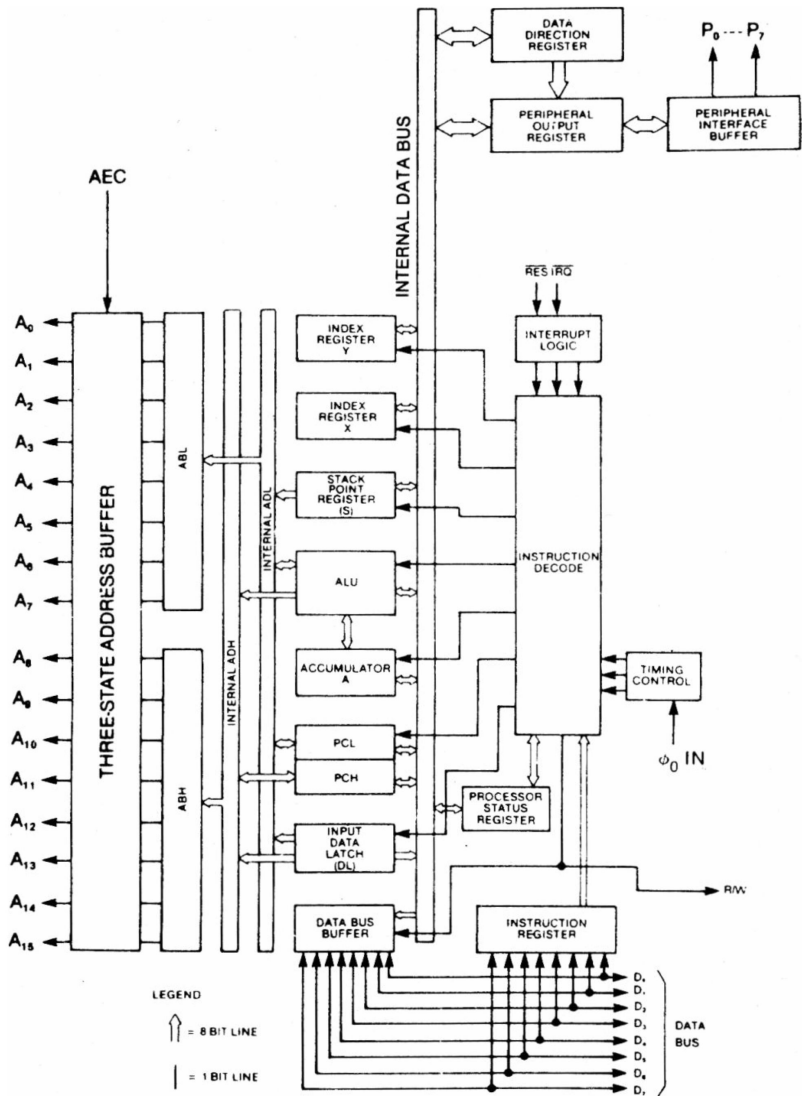



Figura 5.3 Architettura interna della CPU 7501

contiene 6 registri a disposizione del programmatore. Ogni registro e' formato da 8 bit (tranne il PROGRAM COUNTER che ha 16 bit), ed ha la caratteristica di poter immagazzinare o fornire alla propria uscita un dato di 8 bit. I registri non elaborano le informazioni, ma semplicemente "ricordano" lo stato degli 8 bit di cui sono composti. I registri sono i seguenti:

. Index register Y: REGISTRO INDICE Y
. Index register X: REGISTRO INDICE X
. Stack pointer register: STACK POINTER
. Accumulator: ACCUMULATORE
. PCL: meta' del PROGRAM COUNTER, gli 8 bit meno significativi
. PCH: meta' del PROGRAM COUNTER, gli 8 bit piu' significativi
. Processor status register: registro che contiene 7 bit; ognuno di questi bit, chiamati in gergo FLAG, ha un significato che riflette lo stato interno della CPU; esso dipende dalle istruzioni eseguite o dal risultato dell'ultima operazione. Dei FLAG parleremo piu' dettagliatamente nel seguito di questo paragrafo.

Tutti i registri sono collegati ad un BUS DATI interno, e attraverso questo BUS (insieme di 8 fili che corrispondono ciascuno a un bit) possono caricare o scaricare dati. Ogni registro risulta collegato anche a una parte della CPU indicata nello schema come INSTRUCTION DECODE (decodifica dell'istruzione): questo blocco contiene l'unita' di controllo della CPU, unita' che controlla tutte le abilitazioni dei registri (puo' comandare a un registro di caricare il dato presente sul BUS DATI interno, o di presentare su tale BUS il proprio contenuto) e l'ALU (unita' aritmetico-logica). La decodifica dell'istruzione e' la parte piu' complessa di tutta la CPU: la sua funzione e' quella di generare una sequenza di abilitazioni dei registri in modo da eseguire l'istruzione specificata dal programma (il codice dell'istruzione da eseguire viene posto nell'INSTRUCTION REGISTER, che lo presenta al blocco INSTRUCTION DECODE, come puoi osservare nella parte in basso a destra dello schema).

Abbiamo nominato l'ALU; puoi osservare nello schema che tale unita' e' collegata al BUS DATI interno, all'ACCUMULATORE, e ai due BUS interni ADL e ADH, che sono la parte bassa (ADL, 8 bit meno significativi) e parte alta (ADH, 8 bit piu' significativi) del BUS

INDIRIZZI interno. L'ALU e' in grado di eseguire operazioni aritmetico-logiche tra i vari BUS a cui e' collegata. Ad esempio l'ALU e' in grado di eseguire la somma tra il contenuto del BUS DATI interno e l'accumulatore, e memorizzare il risultato. Essa e' in grado poi di presentare il risultato memorizzato sul bus dati; tutte le operazioni che l'ALU esegue sono comandate dalla sezione INSTRUCTION DECODE.
Vediamo ora come la CPU esegue un'istruzione, ad esempio l'istruzione:
LDA #$24
che si trova nei byte di memoria $1000 e $1001:

```
$1000 = $A9
$1001 = $24
```

L'istruzione LDA # produce l'effetto di caricare nell'accumulatore il dato che segue. Il codice di questa istruzione e' 10101001 binario, cioe' $A9.
Il PROGRAM COUNTER (che da ora chiameremo P.C.) contiene l'indirizzo dell'istruzione che deve essere eseguita, cioe' nel nostro caso $1000. Il contenuto del P.C. viene presentato sul BUS INDIRIZZI, si attende una transizione di Φ0 in modo che la memoria abbia il tempo di fornire il dato richiesto, e si carica il contenuto del BUS DATI esterno (in basso nel disegno) nel registro delle istruzioni. Nel nostro caso tale registro contiene il contenuto del byte $1000, cioe' $A9. INSTRUCTION DECODE (da ora I.D.) riconosce che si tratta dell'istruzione LDA #, ed esegue tutte le abilitazioni necessarie, in particolare:

. PCL (che contiene $00) scarica su DATA BUS interno.

. ALU incrementa DATA BUS ($00) e presenta il risultato ($01) su ADL interno.

. PCL carica da ADL ($01): e' stato incrementato il PROGRAM COUNTER.

. Il BUS indirizzi contiene ora $1001. I.D. attende la transizione di Φ0 per accedere all'indirizzo di memoria specificato.

. Passata la transizione I.D. disabilita l'uscita dell'ALU su ADL e l'uscita di PCL sul BUS DATI interno, abilita DATA BUS BUFFER a presentare sul BUS DATI interno il contenuto del BUS DATI esterno ($24, il contenuto del byte $1001).

. I.D. abilita l'accumulatore a caricare dal BUS DATI interno ($24). E' stato caricato $24 nell'accumulatore.

. I.D. disabilita l'uscita dal DATA BUS BUFFER, abilita PCL ($01) a scaricare sul BUS DATI interno e l'ALU incrementa il contenuto del BUS DATI interno, presentando il risultato ($02) su ADL.

. PCL viene abilitato a caricare da ADL, e intanto viene effettuato un accesso alla locazione $1002 (si attende una transizione di Φ0) per leggere il codice dell'istruzione successiva.

Non ti spaventare se hai trovato difficile seguire tutte queste operazioni: abbiamo affrontato un argomento abbastanza complicato. Non e' essenziale capire perfettamente come funziona dentro la CPU per usarla, tuttavia pensiamo sia utile avere almeno un'idea di come si svolgono le operazioni all'interno di questa "scatola nera". Puoi soffermarti su questi argomenti in una seconda lettura: ti consigliamo di seguire sulla Figura 5.3 tutte le operazioni, se vuoi capire bene l'esempio appena spiegato: in seguito puoi anche provare per esercizio (e per divertimento) a far eseguire alla CPU qualche altra istruzione.

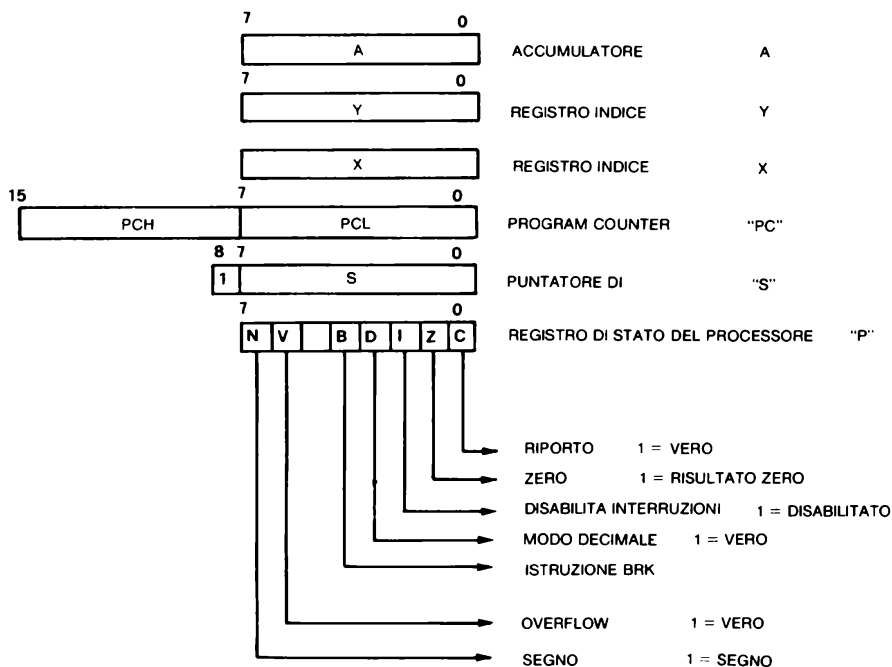



Figura 5.4 Registri della CPU 7501 a disposizione del programmatore

Soffermiamo ora l'attenzione sui FLAG. Abbiamo accennato che vi e' un registro a 8 bit chiamato STATO DEL PROCESSORE che contiene 7 FLAG. Un FLAG e' un bit che ha un certo significato logico. FLAG in inglese vuol dire "bandiera": e' come una bandiera che si alza ad indicare che un determinato evento si e' verificato.
Nella Figura 5.4 trovi riassunti i nomi dei registri della 7501. Puoi osservare come sia organizzato il REGISTRO DI STATO:

. BIT 7 = N. Flag di SEGNO. Questo bit e' a 1 se il risultato dell'operazione e' negativo. La 7501 tratta i numeri negativi in COMPLEMENTO A DUE. Secondo tale rappresentazione il bit piu' significativo rappresenta il segno, (0=positivo, 1=negativo). I numeri negativi sono rappresentati come "parte che manca al valore assoluto del numero per arrivare a 256" (256 nei numeri a 8 bit; 65536 nei numeri a 16 bit; 2^n nei numeri a n bit). Prendiamo il numero -12. Quanto manca da 12 per arrivare a 256? Manca 244; 244 si rappresenta in binario puro 11110100. Questa e' anche la rappresentazione di -12 in complemento a 2. Tale rappresentazione a prima vista appare assai scomoda, ma e' in realta' la piu' efficiente dal punto di vista dell'elaborazione. Il FLAG DI SEGNO riporta lo stato del bit piu' significativo del risultato. Il suo valore viene aggiornato dopo le istruzioni di somma, sottrazione, incremento, decremento, caricamento da memoria, confronto, rotazione di bit, scorrimento di bit, trasferimento da registro a registro, ecc. La tabella nell'Appendice A mostra nella sesta colonna da destra quali sono le istruzioni che alterano lo stato di questo FLAG.
. BIT 6 = V. Overflow. Questo bit indica se c'e' stato traboccamento nelle operazioni in complemento a 2, cioe' se il risultato e' troppo grande (>127) o troppo piccolo (<-128) per essere rappresentato in complemento a 2 con 8 bit. In questo caso il risultato e' sbagliato. Nell'aritmetica in valore assoluto e nei byte meno significativi dei numeri in complemento a 2 che occupano piu' di un byte, lo stato di questo bit non ha nessun significato. Solamente somma e sottrazione alterano lo stato di questo FLAG, oltre all'istruzione BIT (in questo caso V assume un significato diverso, e indica lo stato del bit 6 del risultato), e CLV che pone a 0 lo stato di questo bit.

. BIT 5 = non utilizzato
. BIT 4 = B. Istruzione BRK. L'istruzione BRK e'

molto particolare: quando viene incontrata la CPU esegue le stesse operazioni di quando riceve un interrupt (vedi Paragrafi 5.5 e 4.3), con la differenza che il registro di stato salvato nello stack contiene 1 nella posizione del bit di BREAK. Normalmente nella programmazione non viene usato. Poiche' non esistono istruzioni che permettono di vedere lo stato di questo FLAG nella CPU, bisogna prelevarlo dallo stack con una piccola routine. Puoi vedere questa routine disassemblando il sistema operativo, partendo dall'indirizzo $CE00. Se il FLAG di BRK e' a 1 il SISTEMA OPERATIVO fa un salto indiretto a $0316, se no a $0314.

. BIT 3 = D. Modo decimale. Abbiamo visto nel precedente paragrafo che rappresentando i numeri col modo BCD (decimale a codifica binaria) le operazioni aritmetiche devono essere eseguite con un criterio diverso rispetto al binario puro. Questo FLAG, che e' controllato dal programmatore, informa l'ALU che i conti vanno fatti in BCD. L'istruzione SED attiva il modo BCD, e CLD lo disattiva.

. BIT 2 = I. Disabilita le interruzioni: questo bit e' sotto il controllo diretto del programmatore, cioe' non cambia in base al risultato di operazioni; lo si puo' porre a 1 o a 0, l'istruzione SEI lo pone a 1, l'istruzione CLI lo pone a 0. Come tutti gli altri FLAG il suo valore viene aggiornato anche dopo le istruzioni PLP e RTI, che prelevano lo stato del processore dallo STACK.

. BIT 1 = Z. Risultato zero. Questo e' uno dei FLAG piu' usati nei programmi in ASSEMBLER: indica se tutti gli 8 bit del risultato sono a 0, e solo in tal caso Z vale 1. Come per il FLAG N, anche il FLAG Z viene modificato da numerose istruzioni, come puoi osservare nella quinta colonna da destra dell'Appendice A.

. BIT 0 = C. Carry, o riporto o prestito. Questo FLAG viene alterato dal risultato di diverse operazioni, e talvolta con significati diversi. Nella somma indica se il risultato ha superato 255, e nella sottrazione indica se c'e' stato prestito. Puoi vedere quali operazioni interessano il FLAG C nella quarta colonna da destra della tabella nell'Appendice A.

Abbiamo nominato i termini STACK e STACK POINTER, senza descrivere il loro significato: lo STACK e' un'area di memoria riservata alla CPU dove la CPU pone delle informazioni in maniera temporanea. STACK vuol dire "catasta", e il suo funzionamento e' paragonabile a quello di una catasta o, piu' elegantemente,

di una struttura di dati di tipo LIFO (Last In, First Out, cioe' l'ultimo dato entrato e' il primo che esce). Nella 7501 lo STACK occupa le locazioni da $100 a $1FF, cioe' la pagina 1 della memoria. Lo STACK POINTER e' un registro della CPU di 8 bit, che indica la parte meno significativa del primo byte libero dello STACK (la parte piu' significativa e' sempre $01). All'accensione lo STACK POINTER viene posto a $FF. Ogni volta che si spinge un dato nello STACK (PUSH) il dato viene posto all'indirizzo specificato dallo STACK POINTER, dopodiche' lo STACK POINTER viene decrementato.

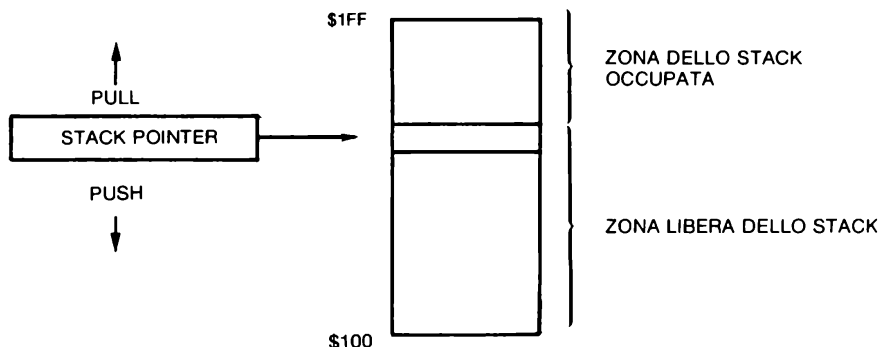



Figura 5.5 STACK e STACK POINTER

Nel prelevare un dato dallo STACK (PULL) viene prima incrementato lo STACK POINTER e poi prelevato il dato. Lo STACK e' molto importante per il funzionamento del calcolatore: l'istruzione JSR (salta alla subroutine), ad esempio, usa lo STACK per memorizzare l'indirizzo dove tornare al momento dell'istruzione RTS, ed anche l'INTERRUPT fa uso dello STACK per poter riprendere il programma alla fine della routine di servizio dell'interrupt. Vi sono diverse istruzioni che fanno uso dello STACK, come PHP, PLP, PHA, PLA, RTS, RTI, BRK; inoltre lo STACK POINTER puo' essere trasferito nel registro X (TSX) e viceversa (TXS). Vedi il Paragrafo 5.4 per la descrizione del funzionamento di queste istruzioni.
Abbiamo gia' visto la funzione dei registri STATO del processore, STACK POINTER, PROGRAM COUNTER. Vediamo ora l'ACCUMULATORE.
Come puoi notare dalla Figura 5.3 l'ACCUMULATORE e' collegato direttamente all'ALU. E' questo il registro

a cui fa riferimento gran parte delle istruzioni aritmetico-logiche. Sull'accumulatore e' possibile effettuare scorrimento di bit, operazioni logiche AND, OR, EOR (Exclusive OR), somma, sottrazione.
I due registri indice X e Y sono usati come indici negli indirizzamenti indicizzati, come puoi meglio osservare nel prossimo paragrafo.

5.3 I MODI DI INDIRIZZAMENTO

Nella descrizione della 7501 abbiamo detto che tale CPU ha molti modi di indirizzamento, e molto potenti; ora passiamo a descriverli.
Il modo di indirizzamento in una CPU e' il modo con cui e' possibile ricavare l'indirizzo dell'operando. Poiche' ogni istruzione ha lo scopo di produrre qualche effetto su qualche operando (registro della CPU o cella di memoria), possiamo dedurre che ogni istruzione (anche NOP, che fa cambiare il PROGRAM COUNTER per eseguire l'istruzione seguente) usa un modo di indirizzamento. Nella prima riga dell'Appendice A sono riportati tutti i modi di indirizzamento della 7501. VEDIAMOLI UNO PER UNO.

. IMMEDIATO. L'operando segue l'istruzione. Si indica con il segno # prima dell'operando. Ad esempio:

LDA #$24

si codifica in memoria cosi':

$A9 $24

Nell'accumulatore viene posto il numero $24.

. ASSOLUTO. L'operando e' contenuto in memoria all'indirizzo specificato dai due byte seguenti il codice dell'istruzione:

LDA $134F

si codifica in memoria cosi':

$AD $4F $13

Nell'accumulatore viene posto il numero contenuto nel byte di indirizzo $134F (prima il byte basso e poi quello alto).

. PAGINA ZERO. L'operando e' contenuto in memoria all'indirizzo della pagina 0 specificato dal byte che segue l'istruzione:

LDA $24

si codifica in memoria cosi':

$A5 $24

A differenza dell'immediato, nell'accumulatore viene posto il byte il cui indirizzo e' contenuto nel byte di indirizzo $0024 (e non il numero $24).

. ACCUMULATORE. L'operando e' dentro la CPU, e precisamente nell'accumulatore:

LSR

si codifica in memoria cosi':

$4A

Viene fatto scorrere verso destra il contenuto dell'accumulatore.

. IMPLICATO. L'operando e' sottinteso dal tipo di istruzione:

CLC

si codifica in memoria cosi':

$18

Azzera il CARRY. L'operando e' palesemente il CARRY.

. ASSOLUTO,X. L'operando e' contenuto in memoria, e l'indirizzo si ricava sommando ai due byte, che seguono il codice dell'istruzione, il contenuto del registo X:

LDA $103D,X

si codifica in memoria cosi':

$BD $3D $10

Supponiamo che il registro X contenga $D1: $103D e' sommato a $D1, ottenendo $110E. L'operando e' contenuto all'indirizzo $110E.

. ASSOLUTO,Y. Opera come ASSOLUTO,X, ma anziche' il registro X usa il registro Y:

LDA $103D,Y

si codifica in memoria cosi':

$B9 $3D $10.

. PAGINA 0,X L'operando si trova in pagina 0, all'indirizzo ottenuto sommando il byte seguente l'istruzione e il registro X:

LDA $F0,X

si codifica in memoria cosi':

$B5 $F0

Se X contiene $32, nell'accumulatore viene posto il byte contenuto nella cella $0022 ($F0 + $32 =$122, ma l'operando viene prelevato sempre dalla pagina 0).

. PAGINA 0,Y. Opera come PAGINA 0,X, ma usa il registro Y. E' implementato solo con le istruzioni LDX e STX:

LDX $20,Y

si codifica in memoria cosi':

$B6 $20.

. INDIRETTO. L'istruzione e' seguita da due byte che indicano l'indirizzo del puntatore dove si trova l'indirizzo dell'operando:

JMP ($2000)

si codifica in memoria cosi':

```
$6C $00 $20
```

Produce un salto non alla cella $2000, ma all'indirizzo specificato nelle celle $2000 e $2001. Se tali celle contengono i numeri $12 e $34 rispettivamente, la CPU salta all'indirizzo $3412 (il byte piu' significativo e' sempre dopo quello meno significativo).

. (INDIRETTO),Y. Si chiama INDIRETTO INDICIZZATO, ed e' uno dei modi piu' potenti. L'istruzione e' seguita da un byte, che indica l'indirizzo in pagina 0 del puntatore al dato. All'indirizzo contenuto nel puntatore va sommato il contenuto del registro Y per ricavare l'indirizzo dell'operando:

```
LDA ($03),Y
```

si codifica in memoria cosi':

```
$B1 $03
```

Se le celle $0003 e $0004 contengono rispettivamente $14 e $2E e il registro Y contiene $F1, l'indirizzo che risulta e' il seguente: $2E14 + $F1 = $2F05. Viene caricato l'accumulatore col contenuto della cella $2F05.

. (INDIRETTO,X). Si chiama INDICIZZATO INDIRETTO, e differisce dal precedente. Il contenuto del registro X e' sommato al byte seguente l'istruzione, il risultato indica l'indirizzo di un puntatore in pagina 0. L'operando si trova all'indirizzo specificato dal puntatore suddetto:

```
LDA ($F0,X)
```

si codifica in memoria cosi':

```
$A1 $F0
```

Se X contiene $31 viene sommato $F0 con $31, ottenendo $121. E' considerata la parte meno significativa, e l'indirizzo dell'operando e' prelevato dalle celle $21 e $22.

. RELATIVO. Questo indirizzamento e' usato per tutte le istruzioni di salto condizionato, o BRANCH. L'operando, che segue l'istruzione, e' un byte considerato come un numero in complemento a due (puo' variare da -128 a +127). L'operando e' sommato al PROGRAM COUNTER e il risultato ottenuto e' immagazzinato nel PROGRAM COUNTER. L'operando indica lo "spiazzamento" da fornire al PROGRAM COUNTER nel caso in cui si verifica la condizione di salto.

```
BCC $09
```

si codifica in memoria cosi':

```
$90 $09
```

e produce l'effetto seguente: se il CARRY e a 1 prepara il P.C. per eseguire l'istruzione che segue, se invece CARRY = 0 al P.C viene sommato 9 rispetto sempre all'istruzione che segue, e il programma riprende dal nono byte che segue l'istruzione dopo BRANCH (ma non necessariamente dalla nona istruzione, perche' una istruzione puo' occupare uno, due o tre byte). Nota che usando il MONITOR devi fornire l'indirizzo assoluto, e il MONITOR calcola l'indirizzo relativo. Il vantaggio di questo tipo di indirizzamento e' che puoi trasferire il programma in un'altra zona della memoria senza cambiare gli indirizzi dei salti.

Abbiamo analizzato tutti i modi possibili di indirizzamento della CPU 7501. Non occorre impararli tutti a memoria subito, ma ti consigliamo di rileggere quelli che non hai capito prima di avventurarti nella scrittura di programmi in linguaggio ASSEMBLER.

5.4 IL SET DI ISTRUZIONI

Nell'esposizione del completo set di istruzioni del microprocessore 7501, distingueremo cinque categorie principali:

1) .trasferimento dati,
2) .logico matematiche,
3) .controllo del flusso,
4) .manipolazione dei flag,
5) .uso dello stack.

.1): Consentono di trasferire dati di 8 bit da un registro all'altro, da un registro alla memoria o viceversa.
.2): Permettono al microprocessore di eseguire operazioni aritmetiche (piu' e meno), operazioni logiche (AND, OR, EOR (exclusive OR)), operazioni di scorrimento (shift e rotazioni), incremento e decremento.
.3): Sono usate per i salti condizionati, salti non condizionati, salti a subroutine e gestione dell'interrupt.
.4): Consentono di agire sul registro di stato del processore.
.5): Permettono di introdurre ed estrarre dati dallo stack, senza doverti preoccupare della gestione dello stack pointer.

Nel corso dell'esposizione compaiono alcuni simboli di cui riportiamo il significato:

A: accumulatore
M: cella di memoria (byte di memoria)
P: registro di stato
S: stack pointer
X: registro X
Y: registro Y
DATO: dato specificato
PC: program counter
STACK: l'ultima cella dello stack
=: assegnamento
Λ: AND logico
V: OR logico
$\underline{V}$: EOR (OR esclusivo)
$\overline{C}$: NOT CARRY
(...): contenuto di ...
BN: bit N
MN: bit N della cella (byte) di memoria specificata.
V: FLAG di overflow
N: FLAG di segno
I: FLAG di interrupt
D: FLAG decimale
C: FLAG di carry
Z: FLAG di zero

5.4.1 LE OPERAZIONI LOGICHE

Prima di illustrare le istruzioni del set della CPU 7501, vorremmo spendere qualche parola riguardo alle operazioni logiche AND OR ed EOR (exclusive OR).
Hai gia' incontrato le prime due in BASIC e le hai usate per fare di piu' condizioni una sola condizione, ad esempio:

```
IF (A AND Z<1) OR A$="PIPPO" THEN ...
```

l'istruzione dopo il THEN viene eseguita se sono vere entrambe le prime due condizioni o se e' vera la terza: cioe' se e' vera la condizione composta.
Ma le operazioni logiche AND e OR sono piu' generali di quanto non possa sembrare dall'esempio appena fatto: esse possono agire, infatti, anche tra numeri interi, oltre che tra condizioni vere o false.

Cominciamo a definire le operazioni logiche tra bit:

AND	OR	EOR
0 AND 0 = 0	0 OR 0 = 0	0 EOR 0 = 0
0 AND 1 = 0	0 OR 1 = 1	0 EOR 1 = 1
1 AND 0 = 0	1 OR 0 = 1	1 EOR 0 = 1
1 AND 1 = 1	1 OR 1 = 1	1 EOR 1 = 0

Il risultato della AND tra due bit e' quindi uguale a 1 solo se il primo E il secondo bit sono uguali a 1; il risultato della OR tra due bit e' uguale a uno se lo sono il primo O il secondo (o entrambi); il risultato della EOR tra due bit e' uguale a 1 se lo sono il primo O il secondo (ma non entrambi).
Estendere ora il significato dell'operazione a numeri interi di X bit e' semplice: il bit N del risultato di un'operazione logica tra byte e' uguale al risultato dell'operazione logica tra i bit corrispondenti dell'operando; ad esempio:

2 AND 3 = 2
infatti:
00000010 AND
00000011 =
00000010 cioe' 2

Le AND e OR del BASIC sono proprio le operazioni logiche che abbiamo descritto.
Prova a chiedere al tuo COMMODORE 16:

PRINT 2 AND 3

e vedrai che il risultato sara' corretto. Queste operazioni vanno bene anche per legare tra di loro condizioni perche' il COMMODORE 16 assegna ad un'espressione vera il valore -1 (in complemento a due su 16 bit = 16 bit a 1) e ad una falsa il valore 0 (16 bit a 0) (prova a scrivere PRINT 2>7 o PRINT 2<7): quando deve combinare tra loro due condizioni con la funzione AND il risultato e' -1 (cioe' vero) solo se entrambe le condizioni valgono -1 (cioe' sono vere); se l'operazione e' la OR il risultato e' -1 se almeno una delle due vale -1.
A questo punto sorge spontanea una domanda: perche' nel set di istruzioni di un microprocessore appaiono queste strane operazioni mentre non compaiono operazioni piu' usuali come la moltiplicazione e la divisione? Perche' queste operazioni ti permettono di leggere o alterare un singolo bit o un gruppo di bit

di un byte, cosa questa assai piu' importante, in molti casi, di una moltiplicazione o una divisione (che si possono comunque ottenere grazie a un opportuno algoritmo). Ad esempio: vuoi sapere quanto vale il bit 3 di un certo byte? Basta fare la AND tra questo byte e 8 (2^3): se il risultato e' 0 il bit vale 0, se e' 8 il bit e' a 1. Altro esempio: vuoi porre a 1 il bit 5? Fai la OR con 32 (2^5). Se lo volevi a 0? Fai la AND con 223 (255-32 cioe' un numero che ha tutti i bit a uno tranne il quinto).La EOR a cosa serve? Serve a cambiare il valore di uno o piu' bit. Ad esempio 01010011 EOR 00001111 = 01011100: questa operazione ha cambiato il valore dei bit 3-0 del primo operando (cioe' quelli che valgono 1 nel secondo operando). Se fai una EOR tra il contenuto di un byte e $FF "neghi" il contenuto del byte: cioe' i bit che valevano 1 valgono 0 e viceversa.
Ti proponiamo il programma AND&OR che genera operazioni logiche casuali tra numeri binari di 8 bit. Dopo aver calcolato la risposta premi un tasto e il tuo calcolatore ti dara' la conferma.

```
0 REM AND&OR
10 A=INT(RND(0)*256):B=INT(RND(0)*256)
11 OP=INT(RND(0)*2)
20 PRINTCHR$(147)
30 FORI=0TO7
40 IFAAND2↑(7-I)THENCHAR,4+I,3,"●":
   ELSECHAR,4+I,3,"o"
50 NEXTI:CHAR,12,3,STR$(A)
60 IFOPTHENCHAR,1,4,"OR":ELSECHAR,0,4,"AND"
70 FORI=0TO7
80 IF B AND2↑(7-I)THENCHAR,4+I,4,"●":
   ELSECHAR,4+I,4,"o"
90 NEXTI:CHAR,12,4,STR$(B)
100 CHAR,2,6,"="
110 IFOPTHENR=AORB:ELSER=AANDB
115 GETKEYA$
120 FORI=0TO7
130 IFRAND2↑(7-I)THENCHAR,4+I,6,"●":
    ELSECHAR,4+I,6,"o"
140 NEXTI:CHAR,12,6,STR$(R)
150 GETKEYA$:RUN
```

5.4.2 ISTRUZIONI DI TRASFERIMENTO

LDA: LoaD Accumulator A=DATO

Il dato specificato viene posto nell'accumulatore.
INDIRIZZAMENTO: immediato - assoluto - pagina 0 - indicizzato X e Y - pagina 0 indicizzato X - indiretto indicizzato - indicizzato indiretto.
FLAG MODIFICATI: zero e segno.

LDX: LoaD X register X=DATO

Il dato specificato viene posto nel registro X.
INDIRIZZAMENTO: immediato - assoluto - pagina 0 - indicizzato Y - pagina 0 indicizzato Y.
FLAG MODIFICATI: zero e segno.

LDY: LoaD Y register Y=DATO

Il dato specificato viene posto nel registro Y.
INDIRIZZAMENTO: immediato - assoluto - pagina 0 - indicizzato X - pagina 0 indicizzato X.
FLAG MODIFICATI: zero e segno.

STA: STore Accumulator M=(A)

Il contenuto dell'accumulatore viene posto nella cella di memoria di indirizzo specificato. Il contenuto dell'accumulatore non viene cambiato.
INDIRIZZAMENTO: assoluto - pagina 0 - indicizzato X e Y - pagina 0 indicizzato X - indiretto indicizzato - indicizzato indiretto.
FLAG MODIFICATI: nessuno.

STX: STore X register M=(X)

Il contenuto del registro X viene posto nella cella di memoria di indirizzo specificato. Il contenuto del registro X non viene cambiato.
INDIRIZZAMENTO: assoluto - pagina 0 - pagina 0 indicizzato Y.
FLAG MODIFICATI: nessuno.

==

STY: STore Y register M=(Y)

Il contenuto del registro Y viene posto nella cella di memoria di indirizzo specificato. Il contenuto del registro Y non viene cambiato.
INDIRIZZAMENTO: assoluto - pagina 0 - pagina 0 indicizzato X.
FLAG MODIFICATI: nessuno.

==

TAX: Transfer Accumulator to X register X=(A)

Il contenuto dell'accumulatore viene posto nel registro X. Il contenuto dell'accumulatore non viene cambiato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

==

TAY: Transfer Accumulator to Y register Y=(A)

Il contenuto dell'accumulatore viene posto nel registro Y. Il contenuto dell'accumulatore non viene cambiato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

==

TSX: Transfer Stack pointer to X register X=(S)

Il contenuto dello stack pointer viene posto nel registro X. Il contenuto dello stack pointer non viene cambiato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

==

TXA: Transfer X register to Accumulator A=(X)

Il contenuto del registro X viene posto nell'accumulatore. Il contenuto del registro X non viene cambiato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

==

TXS: Transfer X register to Stack pointer S=(X)

Il contenuto del registro X viene posto nello stack pointer. Il contenuto del registro X non viene cambiato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: nessuno.

===

TYA: Transfer Y register to Accumulator   A=(Y)

Il contenuto del registro Y viene posto nell'accumulatore. Il contenuto del registro Y non viene cambiato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

5.4.3 OPERAZIONI LOGICO MATEMATICHE.

===

ADC: ADd with Carry   A=(A)+DATO+C

Somma il contenuto dell'accumulatore con il dato specificato e il carry. Il risultato viene posto nell'accumulatore.
INDIRIZZAMENTO: immediato - assoluto - pagina 0 - indicizzato X e Y - pagina 0 indicizzato X - indiretto indicizzato - indicizzato indiretto.
FLAG MODIFICATI: carry, zero, overflow e segno.

===

AND: AND   A=(A)∧DATO

Esegue l'AND logico tra l'accumulatore e il dato specificato. Il risultato viene posto nell'accumulatore.
INDIRIZZAMENTO: immediato - assoluto - pagina 0 - pagina 0 indicizzato X - indicizzato X e Y - indiretto indicizzato - indicizzato indiretto.
FLAG MODIFICATI: zero e segno.

===

ASL: Arithmetic Shift Left
C=B7=B6=B5=B4=B3=B2=B1=B0=0

Sposta il contenuto dell'accumulatore o della cella di memoria specificata di una posizione bit verso sinistra. Il contenuto del bit 0 diventa 0 e il contenuto del bit 7 viene posto nel flag di carry. Questa opera-

zione equivale ad eseguire una moltiplicazione per 2.
INDIRIZZAMENTO: accumulatore - assoluto - pagina 0 - indicizzato X - pagina 0 indicizzato X.
FLAG MODIFICATI: carry, zero e segno.

===

BIT: test BITs in memory

(A)∧(M) - N=(M7) - V=(M6)

Esegue la AND logica tra il contenuto dell'accumulatore e il contenuto della cella di memoria specificata. Il risultato non viene posto da nessuna parte ma viene alterato il flag di zero. Pone il contenuto del bit 7 della cella di memoria specificata nel flag di segno e il contenuto del bit 6 nel flag di overflow.
INDIRIZZAMENTO: assoluto - pagina 0.
FLAG MODIFICATI: zero, overflow e segno.

===

CMP: CoMPare with accumulator (A)-DATO

Sottrae al contenuto dell'accumulatore il dato specificato. Il risultato non viene posto da nessuna parte ma vengono alterati i flag di carry, zero e segno (C=1 indica A>=DATO, Z=1 indica A=DATO).
INDIRIZZAMENTO: immediato - assoluto - pagina 0 - indicizzato X e Y - pagina 0 indicizzato X - indiretto indicizzato - indicizzato indiretto.
FLAG MODIFICATI: carry, zero e segno.

===

CPX: ComPare with X register (X)-DATO

Sottrae al contenuto del registro X il dato specificato. Il risultato non viene posto da nessuna parte ma vengono alterati i flag di carry, zero e segno (C=1 indica A>=DATO, Z=1 indica A=DATO).
INDIRIZZAMENTO: immediato - assoluto - pagina 0
FLAG MODIFICATI: carry, zero e segno.

===

CPY: ComPare with Y register (Y)-DATO

Sottrae al contenuto del registro Y il dato specificato. Il risultato non viene posto da nessuna parte ma vengono alterati i flag di carry, zero e segno (C=1 indica A>=DATO, Z=1 indica A=DATO).

INDIRIZZAMENTO: immediato - assoluto - pagina 0
FLAG MODIFICATI: carry, zero e segno.

DEC: DECrement M=(M)-1

Decrementa il contenuto della cella di memoria specificata. Il risultato viene posto nella cella stessa.
INDIRIZZAMENTO: assoluto - pagina 0 - indicizzato X - pagina 0 indicizzato X.
FLAG MODIFICATI: zero e segno.

DEX: DEcrement X register X=(X)-1

Decrementa il contenuto del registro X. Il risultato viene posto nello stesso registro.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

DEY: DEcrement Y register Y=(Y)-1

Decrementa il contenuto del registro Y. Il risultato viene posto nello stesso registro.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

EOR: Exclusive OR A=(A)⊻DATO

Esegue l'OR esclusivo tra l'accumulatore e il dato specificato. Il risultato viene posto nell'accumulatore.
INDIRIZZAMENTO: immediato - assoluto - pagina 0 - pagina 0 indicizzato X - indicizzato X e Y - indiretto indicizzato - indicizzato indiretto.
FLAG MODIFICATI: zero e segno.

INC: INCrement M=(M)+1

Incrementa il contenuto della cella di memoria specificata. Il risultato viene posto nella cella stessa.
INDIRIZZAMENTO: assoluto - pagina 0 - indicizzato X - pagina 0 indicizzato X.
FLAG MODIFICATI: zero e segno.

==

INX: INcrement X register X=(X)+1

Incrementa il contenuto del registro X. Il risultato viene posto nello stesso registro.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

==

INY: INcrement Y register Y=(Y)+1

Incrementa il contenuto del registro Y. Il risultato viene posto nello stesso registro.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

==

LSR: Logic Shift Right
C=B0=B1=B2=B3=B4=B5=B6=B7=0

Sposta il contenuto dell'accumulatore o della cella di memoria specificata di una posizione bit verso destra. Il contenuto del bit 7 diventa 0 e il contenuto del bit 0 viene posto nel flag di carry. Questa operazione equivale ad eseguire una divisione intera per 2.
INDIRIZZAMENTO: accumulatore - assoluto - pagina 0 - indicizzato X - pagina 0 indicizzato X.
FLAG MODIFICATI: carry, zero e segno.

==

ORA: OR Accumulator A=(A)VDATO

Esegue l'OR logico tra l'accumulatore e il dato specificato. Il risultato viene posto nell'accumulatore.
INDIRIZZAMENTO: immediato - assoluto - pagina 0 - indicizzato X e Y - pagina 0 indicizzato X - indiretto indicizzato - indicizzato indiretto.
FLAG MODIFICATI: zero e segno.

==

ROL: ROtate Left
C=B7=B6=B5=B4=B3=B2=B1=B0=C

Ruota il contenuto dell'accumulatore o della cella di memoria specificata di una posizione bit verso sinistra. Nel bit 0 viene posto il contenuto del flag di carry, nel quale viene trasferito il contenuto del bit 7.

INDIRIZZAMENTO: accumulatore - assoluto - pagina 0 - indicizzato X - pagina 0 indicizzato X.
FLAG MODIFICATI: carry, zero e segno.

==

ROR: ROtate Right

C=B0=B1=B2=B3=B4=B5=B6=B7=C

Ruota il contenuto dell'accumulatore o della cella di memoria specificata di una posizione bit verso destra. Nel bit 7 viene posto il contenuto del flag di carry, nel quale viene trasferito il contenuto del bit 0.
INDIRIZZAMENTO: accumulatore - assoluto - pagina 0 - indicizzato X - pagina 0 indicizzato X.
FLAG MODIFICATI: carry, zero e segno.

==

SBC: SuBtract with Carry A=(A)-DATO-$\overline{C}$

Sottrae al contenuto dell'accumulatore il dato specificato. Il risultato viene posto nell'accumulatore. Devi usare il carry al contrario di come lo usi per l'istruzione ADC: carry a 0 vuol dire prestito; devi quindi porre a 1 il carry prima di eseguire un'operazione SBC senza prestito. Dopo un operazione SBC si ha: C=1 indica A>=DATO, Z=1 indica A=DATO.
INDIRIZZAMENTO: immediato - assoluto - pagina 0 - indicizzato X e Y - pagina 0 indicizzato X - indiretto indicizzato - indicizzato indiretto.
FLAG MODIFICATI: carry, zero, overflow e segno.

5.4.4 ISTRUZIONI DI CONTROLLO DEL FLUSSO

==

BCC: Branch on Carry Clear

Salta all'indirizzo specificato se il flag di carry contiene 0.
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

==

BCS: Branch on Carry Set

Salta all'indirizzo specificato se il flag di carry contiene 1.

INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

BEQ: Branch on EQual

Salta all'indirizzo specificato se il flag di zero contiene 1 (cioe' se il risultato e' 0).
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

BMI: Branch on MInus

Salta all'indirizzo specificato se il flag di segno contiene 1 (cioe' se il risultato e' negativo).
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

BNE: Branch on Not Equal

Salta all'indirizzo specificato se il flag di zero contiene 0 (cioe' se il risultato e' diverso da 0).
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

BPL: Branch on PLus

Salta all'indirizzo specificato se il flag di segno contiene 0 (cioe' se il risultato e' positivo o uguale a 0).
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

BRK: BReaK

Salva nello stack (PC)+2 e i flag, salta all'indirizzo indicato da $FFFE $FFFF (come per gli interrupt). Per riconoscere se il salto e' stato generato da un interrupt o dall' istruzione BRK devi guardare il flag di break salvato nello stack. BRK viene usato per trovare errori nel programma (come lo STOP del BASIC).
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: break

==

BVC: Branch on oVerflow Clear

Salta all'indirizzo specificato se il flag di overflow contiene 0 (cioe' se non si e' verificato overflow).
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

==

BVS: Branch on oVerflow Set

Salta all'indirizzo specificato se il flag di overflow contiene 1 (cioe' se si e' verificato overflow).
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

==

JMP: JuMP

Salta all'indirizzo specificato.
INDIRIZZAMENTO: assoluto - indiretto.
FLAG MODIFICATI: nessuno.

==

JSR: Jump SubRoutine

Salva nello stack (PC)+2 (l'indirizzo prima dell'istruzione che segue JSR) , salta all'indirizzo specificato.
INDIRIZZAMENTO: assoluto.
FLAG MODIFICATI: nessuno.

==

NOP: No OPeration
Non fa nulla per due cicli di clock. Si usa per cicli di ritardo.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: nessuno.

==

RTI: ReTurn from Interrupt

Estrae dallo stack il registro di stato e il program counter che erano stati salvati da una chiamata ad interrupt o da una istruzione BRK.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: tutti.

==

RTS: ReTurn from Subroutine

Estrae dallo stack il program counter che era stato salvato da una istruzione JSR e lo incrementa di 1.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: nessuno.

5.4.5 ISTRUZIONI SUI FLAG.

==

CLC: CLear Carry C=0

Viene azzerato il flag di carry. Generalmente si usa prima di un'istruzione ADC.
INDIRIZZAMENTO: implicito
FLAG MODIFICATI: carry.

==

CLD: CLear Decimal D=0

Viene azzerato il flag BCD. Quando questo flag contiene 0, l'ALU esegue le operazioni aritmetiche in binario.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: decimale.

==

CLI: CLear Interrupt I=0

Viene azzerato il flag di interrupt. Quando questo flag contiene 0 la CPU risponde alle chiamate d'interrupt.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: interrupt.

==

CLV: CLear oVerflow V=0

Viene azzerato il flag di overflow.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: overflow.

==

SEC: SEt Carry C=1

Viene posto a uno il flag di carry. Generalmente si

usa prima di un'istruzione SBC.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: carry.

==

SED: SEt Decimal D=1

Viene posto a uno il flag decimale. Quando questo flag contiene 1, l'ALU esegue le operazioni aritmetiche in BCD.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: decimale.

==

SEI: SEt Interrupt I=1

Viene posto a uno il flag di interrupt. Quando questo flag contiene 1 la CPU non risponde alle chiamate d'interrupt.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: interrupt.

5.4.6 OPERAZIONI SULLO STACK

==

PHA: PusH Accumulator STACK=(A) ; SP=SP-1

Il contenuto dell'accumulatore viene memorizzato nella cella di indirizzo $100+(SP); lo stack pointer viene decrementato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: nessuno.

==

PHP: PusH Processor status STACK=(P); SP=SP-1

Il contenuto del registro di stato viene memorizzato nella cella di indirizzo $100+(SP); lo stack pointer viene decrementato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: nessuno.

==

PLA: PuLl Accumulator A=(STACK); SP=SP+1

Il contenuto della cella di indirizzo $100+(SP) viene

caricato nell'accumulatore; lo stack pointer viene incrementato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: nessuno.

===

PLP: PuLl Processor status P=(STACK); SP=SP+1

Il contenuto della cella di indirizzo $100+(SP) viene caricato nel registo di stato; lo stack pointer viene incrementato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: tutti.

5.5 LA GESTIONE DELL'INTERRUPT

Abbiamo visto nel Paragrafo 4.3 che la CPU possiede un piedino da cui riceve le chiamate di INTERRUPT. Puoi immaginare un segnale di interrupt come la chiamata HARDWARE di un sottoprogramma; la subroutine cioe' non viene eseguita in risposta a una chiamata da programma, ma a un segnale logico che giunge a un piedino della CPU. Il TED e' l'unico dispositivo collegato alla linea di richiesta delle interruzioni; ogni 50-esimo di secondo la CPU riceve da TED una richiesta di interruzione. Il SISTEMA OPERATIVO usa questo segnale per eseguire alcune operazioni a intervalli di tempo regolari, come aggiornare TI$, o memorizzare eventuali caratteri premuti sulla tastiera.
Il programma che segue, in ASSEMBLER, ti mostra come la routine di interrupt, che puo' essere disabilitata, controlla lo stato della tastiera. Puoi introdurlo in memoria usando il MONITOR.

```
   A 3000 CLI
   A 3001 LDA $C6
   A 3003 STA $0C00
   A 3006 CLC
   A 3007 BCC $3001
```

Per avviarne l'esecuzione puoi scrivere:

```
   G 3000.
```

Vedrai che il primo carattere in alto a sinistra nello schermo cambia in funzione del tasto che premi sulla tastiera; questo fatto parrebbe strano sapendo che la cella di indirizzo $C6 e' una normale cella di memoria in pagina 0, e non un registro di ingresso. Se

arresti l'esecuzione del programma (l'unico modo e' il tasto di RESET, eventualmente STOP-RESET) e cambi l'istruzione in $3000, cioe' cambi CLC in SEI (maschecheri le interruzioni), ti accorgi, avviando nuovamente il programma, che la tastiera non influisce piu' sul quadratino del video. Cio' e' dovuto al fatto che la ROUTINE DI INTERRUPT non viene eseguita quando il FLAG di INTERRUPT e' a 1.
Nel Paragrafo 7.6 riportiamo un sottoprogramma, in linguaggio macchina, che intercetta la routine di servizio dell'INTERRUPT del SISTEMA OPERATIVO.
Quando la CPU risponde a una chiamata di interrupt salva automaticamente nello STACK i seguenti dati:

.gli 8 bit piu' significativi del P.C.
.gli 8 bit meno significativi del P.C.
.il registro di STATO del processore

e produce un salto indiretto a $FFFE.
E' cura del SISTEMA OPERATIVO effettuare tutte le operazioni che si devono eseguire in risposta a un interrupt. Per tornare dalla routine di gestione dell' interrupt esiste l'istruzione RTI, che e' diversa da RTS perche' recupera dallo STACK anche lo stato del processore, oltre a non incrementare il P.C.

5.6 USO DEL COMANDO MONITOR DEL BASIC

La funzione di questo comando BASIC e' quella di portare in ambiente MONITOR il calcolatore. Il MONITOR e' un programma in linguaggio macchina che permette di scrivere facilmente programmi in ASSEMBLER e in linguaggio macchina. Esso comprende un MONITOR di linguaggio macchina, un MINI-ASSEMBLATORE e un DISASSEMBLATORE. I programmi in linguaggio macchina, scritti utilizzando il MONITOR, possono essere utilizzati autonomamente oppure come sottoprogrammi molto veloci per programmi BASIC.

I comandi disponibili da MONITOR sono:

. A ASSEMBLA: assembla un'istruzione all'indirizzo specificato.
. C CONFRONTA: confronta due blocchi della memoria e segnala le differenze.
. D DISASSEMBLA: disassembla il codice della 7501 a partire dall'indirizzo specificato.
. F RIEMPI: riempie la memoria con il numero specificato.

. G ESEGUI: avvia l'esecuzione del programma dall'indirizzo specificato.
. H CERCA: ricerca nella memoria tutte le posizioni di determinati valori di un gruppo di numeri.
. L CARICA: carica un file dal nastro o dal disco.
. M VISUALIZZA MEMORIA: visualizza i valori esadecimali delle locazioni di memoria
R VISUALIZZA REGISTRI: visualizza i registri della 7501, che tu puoi cambiare.
. S SALVA: salva su nastro o su disco un blocco di memoria.
. T TRASFERISCI: trasferisce un blocco di memoria in un'altra posizione.
. V VERIFICA: confronta un blocco della memoria con un file su nastro o su disco.
. X ESCI: esce dal MONITOR e ritorna al BASIC.
. . PUNTO: assembla una riga del codice della 7501.
. > (maggiore di): modifica la memoria.
. ; (punto e virgola): modifica i registri della CPU.

La locazione $7F8 controlla se il MONITOR vede la ROM o la RAM sopra $8000. Se questa locazione contiene 0, il MONITOR visualizza l'interprete BASIC e il KERNAL (routine del SISTEMA OPERATIVO) dopo un comando di disassemblaggio o di stampa, cioe' il contenuto della memoria ROM di indirizzo sopra $8000. Se questa locazione contiene $80, il MONITOR visualizza la RAM che sta sotto l'interprete BASIC e il KERNAL. Cio' non e' particolarmente utile nella macchina originale, ma diventa conveniente per lo sviluppo di programmi in linguaggio macchina per chi possiede l'espansione di memoria da 64K. Nota che la locazione $7F8 non ha alcuna influenza sul comando G. Il comando G avvia l'esecuzione nella mappa di memoria ROM senza tener conto dell'impostazione della locazione $7F8.

Si accede al MONITOR scrivendo:

```
MONITOR
```

la risposta del sistema e' la visualizzazione dei registri della 7501 e il cursore lampeggiante. Il cursore rappresenta il "prompt" che ricorda che il MONITOR e' in attesa dei comandi.

I comandi, che passiamo a descrivere, fanno uso di parametri; tutti i parametri sono da intendere in forma esadecimale, li scriviamo non preceduti dal dolla-

ro, tra un parametro e l'altro si puo' porre uno spazio oppure una virgola.

A introduce una riga di codice ASSEMBLER.

SINTASSI: A <indirizzo> <codice operativo mnemonico> <operando>
<indirizzo>: locazione della memoria dove collocare il codice operativo.
<codice operativo mnemonico>: codice mnemonico dell'istruzione da assemblare, cosi' come e' descritto nel Paragrafo 5.4 e nell'APPENDICE A.
<operando>: operando, quando richiesto, puo' essere di una qualsiasi delle modalita' d'indirizzamento ammesse. Per le modalita' di indirizzamento in pagina zero si deve indicare un numero esadecimale di 2 cifre; ad esempio: LDA $10. Per indirizzi di pagina-non zero vengono richiesti numeri esadecimali di 4 cifre; ad esempio: LDA $1000. Per l'indirizzamento immediato si deve porre il segno # prima del numero a 2 cifre esadecimale che rappresenta l'operando; ad esempio: LDA #$FF.
Alla fine della riga devi premere RETURN. Se nella riga risultano degli errori, viene visualizzato un punto di domanda ad indicare un errore e il cursore si sposta alla riga seguente. Per correggere eventuali errori puoi tornare su con il cursore e modificare la linea in modo da eliminare l'errore.
Qui di seguito listiamo il programma LEGGIJOYASS, che puoi introdurre in memoria ricopiandolo con il MONITOR, oppure che si carica automaticamente in memoria usando il programma BASIC TEST LEGGEJOYASS.

```
   PC  SR AC XR YR SP
; 0000 00 00 00 00 F8

. 3000  A2 FB      LDX #$FB
. 3002  78         SEI
. 3003  8E 08 FF   STX $FF08
. 3006  AD 08 FF   LDA $FF08
. 3009  8E 08 FF   STX $FF08
. 300C  CD 08 FF   CMP $FF08
. 300F  D0 F2      BNE $3003
. 3011  58         CLI
. 3012  29 4F      AND #$4F
. 3014  C9 10      CMP #$10
. 3016  90 02      BCC $301A
. 3018  09 80      ORA #$80
. 301A  29 8F      AND #$8F
. 301C  49 8F      EOR #$8F
```

```
. 301E  AA          TAX
. 301F  A0 FD       LDY #$FD
. 3021  78          SEI
. 3022  8C 08 FF    STY $FF08
. 3025  AD 08 FF    LDA $FF08
. 3028  8C 08 FF    STY $FF08
. 302B  CD 08 FF    CMP $FF08
. 302E  D0 F2       BNE $3022
. 3030  58          CLI
. 3031  29 8F       AND #$8F
. 3033  49 8F       EOR #$8F
. 3035  A8          TAY
. 3036  60          RTS
```

Questa routine ti permette di leggere lo stato dei joystick, per usarli nei tuoi programmi in linguaggio macchina. Come puoi osservare il programma fa uso di un registro di I/O, in particolare il registro $FF08, il cui funzionamento e' stato descritto nel Paragrafo 4.6. Puoi osservare nelle Figura 4.7a e 4.7b come sono collegati i due joystick.
Le istruzioni da $3000 a $3011 caricano nell'accumulatore lo stato dei 5 bit collegati al Joystick 1. Da $3012 a $301E diamo un formato piu' adatto al risultato e lo poniamo nel registro X. Da $301F a $3030 poniamo in A lo stato del joystick 2 e da $3031 a $3036 adattiamo il formato del risultato ponendolo nel registro Y. Alla fine della routine il registro X contiene lo stato del joystick 1, e Y quello del joystick 2.

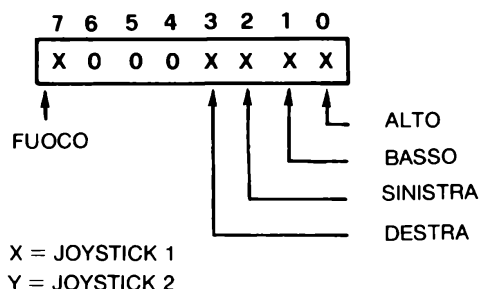



Figura 5.6 Formato dei risultati del sottoprogramma LEGGIJOYASS

Nella Figura 5.6 puoi vedere il significato dei bit dei registri X e Y in relazione alle posizioni dei joystick. I bit a 1 indicano che il tasto e' premuto, o la leva e' spostata. Il programma TEST LEGGIJOYSTICK visualizza lo stato dei registri X e Y alla fine della routine. Il codice del programma inoltre e' contenuto nelle linee DATA, e viene posto in memoria nella linea 20. E' assai importante non sbagliare neanche uno dei dati, poiche' in tal caso il sistema potrebbe arrestarsi, e non riprendere l'esecuzione neanche premendo RESET. Per questo la linea 25 arresta il programma se la somma dei dati contenuti nelle linee DATA differisce dal valore corretto: ricontrolla bene tutti i dati in caso di errore, e non eliminare la linea 25.

```
0 REM TEST ROUTINE LETTURA JOYSTICK IN ASSEMBLER
5 REM LA ROUTINE VA DA 3000H A 3036H COMPRESI
10 POKE56,48:POKE55,0:CLR
20 FORI=12288TO12342:READA:Q=Q+A:POKEI,A:NEXT
25 IFQ<>7420THENSTOP
30 DO
40 SYS12288:A=PEEK(2035):B=PEEK(2036)
50 GOSUB80:PRINT,
60 A=B:GOSUB80:PRINT
70 LOOP
80 FORI=7TO0STEP-1:IFAAND2↑ITHENPRINT"1";:ELSEPRINT"0";
90 NEXT:RETURN
100 DATA162,251,120,142,8,255,173,8,255,142,8
105 DATA255,205,8,255,208,242,88,41,79,201
110 DATA16,144,2,9,128,41,143,73,143,170,160
115 DATA253,120,140,8,255,173,8,255,140,8,255
120 DATA205,8,255,208
130 DATA242,88,41,143,73,143,168,96
```

C confronta due blocchi della memoria e segnala le differenze.

SINTASSI: C <indirizzo inizio blocco 1> <indirizzo fine blocco 1> <indirizzo inizio blocco 2>
Esempio: C 1000 1035 2010
Confronta i contenuti delle celle che vanno da $1000 a $1035 compreso, con quelli delle celle che vanno da

$2010 a $2045 compreso. Se qualche byte differisce, ne viene stampato l'indirizzo.

==

D disassembla il contenuto della cella specificata.

SINTASSI: D <indirizzo iniziale> <indirizzo finale>
<indirizzo iniziale>: numero esadecimale di al massimo 4 cifre che indica l'indirizzo dell'istruzione da disassemblare. Se non fornito vengono disassemblati 20 (decimale) byte a partire dall'ultimo disassemblato.
<indirizzo finale>: 4 cifre esadecimali che indicano a quale indirizzo smettere di disassemblare. Se non fornito vengono disassemblati 20 byte.

==

F riempie una zona di memoria con il numero specificato.

SINTASSI: F <indirizzo iniziale> <indirizzo finale> <dato>
Esempio: F 0C00 0D00 01
riempie la zona di memoria da $0C00 a $0D00 con il numero $01. Tutti i parametri sono obbligatori.

==

G avvia l'esecuzione di un programma contenuto in memoria.

SINTASSI: G <indirizzo>
Il parametro <indirizzo> puo' essere omesso: in tal caso il programma parte dall'indirizzo indicato nel registro PC (vedi il comando R).

==

H ricerca nella memoria le posizioni di determinati byte.

SINTASSI: H <ind. iniz.> <ind. finale> <sequenza di byte> oppure <'sequenza di caratteri>
Cerca nelle locazioni specificate la sequenza di byte specificata.

Esempio: H 1000 2000 1 DE F
cerca se tra $1000 e $2000 e' contenuta la sequenza: $01 $DE $0F. Se tale sequenza e' presente, viene stam¬pato l'indirizzo del primo byte della sequenza, e se e' presente piu' volte, sono stampati tutti gli indi¬rizzi dove inizia la sequenza specificata.
Esempio: H 1000 4000 'COMMODORE
cerca se tra le celle $1000 e $4000 se e' presente la sequenza di byte il cui codice ASCII compone la strin¬ga "COMMODORE". Nota che i caratteri sono preceduti da un apice (SHIFT¬7).

==

L carica un file da nastro o da disco.

SINTASSI: L <"nomefile"> <numero unita'>
Se non si specifica il numero di unita' si considera il registratore a cassetta. Il file viene caricato al suo indirizzo originale (nella stessa posizione in cui era quando e' stato salvato).
Esempio: L
carica da nastro il primo programma che trova.
Esempio: L "ROUTINE",8
carica in memoria da disco il file chiamato ROUTINE.

==

M visualizza i valori esadecimali delle loca¬zioni di memoria specificate.

SINTASSI: M <ind. iniz.> <ind. finale>
Gli operandi hanno lo stesso significato che nel comando D. I byte stampati possono essere modificati passandoci sopra col cursore, cambiandoli e premendo RETURN.
Esempio: M 1000
mostra il contenuto di 96 byte a partire dall'in¬dirizzo 1000.

==

R visualizza il valore dei registri della CPU.

SINTASSI: R
Naturalmente la CPU continua ad eseguire il programma MONITOR e i suoi registri cambiano continuamente. I registri che ti vengono mostrati (e di cui puoi

cambiare il contenuto passandoci sopra e battendo RETURN), sono quelli che saranno caricati al momento dell'avvio del programma, e che sono stati prelevati dalla CPU al momento del BRK. L'istruzione BRK porta al MONITOR, e vengono automaticamente mostrati i contenuti dei registri.

===

S salva su nastro o disco un blocco di memoria.

SINTASSI: S "NOMEPROGRAMMA" <unita'> <indirizzo iniziale> <indirizzo finale>
Tutti i parametri sono obbligatori, tranne il nome del file su cassetta. L'indirizzo finale deve essere uno piu' dell'ultimo byte da salvare:
S "FILE" 08 1001 11F3
Salva su disco il programma di nome FILE a partire dall'indirizzo $1001 fino all'indirizzo $11F2 compreso.

===

T trasferisce segmenti di memoria da una zona a un'altra.

SINTASSI: T <indir. iniz.> <indir. finale> <nuovo indirizzo>
I dati possono essere trasferiti ovunque nella RAM. Purtroppo un errore del sistema non permette di trasferire un segmento di memoria sopra se stesso.
Esempio: T 1400 1600 1401
non produce l'effetto di spostare di una posizione il segmento verso l'alto nella memoria, ma riempie tutta l'area $1401 $1601 con il contenuto della cella $1400, distruggendo il vecchio contenuto. Cio' e' dovuto probabilmente all'errore di eseguire il trasferimento del segmento sempre dal basso verso l'alto, mentre sarebbe corretto trasferire dal basso verso l'alto nei trasferimenti dall'alto verso il basso, e viceversa. Prova per esercizio a scrivere un programma che trasferisce blocchi di memoria, e vedrai meglio quali difficolta' incontra il programmatore. La versione del SISTEMA OPERATIVO a cui ci riferiamo e' la numero 4 per la versione PAL (la cella $FF80 della ROM contiene $84). E' possibile che in versioni successive l'errore sia stato eliminato.

==

V verifica che un file contenga gli stessi dati contenuti in memoria.

SINTASSI: V <"nomefile"> <unita'>

Le regole sono le stesse di L (carica) ma il file non viene caricato in memoria. Se il file differisce dal contenuto della memoria viene stampato il messaggio "VERIFYING ERROR", altrimenti non viene stampato niente.

==

X torna in ambiente BASIC.

SINTASSI: X

Il MONITOR ti pone in grado di alterare celle di memoria e di compiere molte operazioni pericolose. E' importante non alterare il contenuto di celle di memoria di cui non conosci la funzione, soprattutto nella zona di lavoro dell'interprete BASIC e del SISTEMA OPERATIVO ($0000 - $0800) e nell'I/O ($FD00 - $FFFF). Puo' capitare assai facilmente che il sistema impazzisca a causa di operazioni scorrette sulla memoria. Talvolta il comando X non ti porta al BASIC: quando cio' capita e' perche' il sistema e' in umo stato anomalo, da cui puo' uscire solo con un RESET.

CAPITOLO 6

LA GRAFICA

6.1 INTRODUZIONE

In questo capitolo ti mostriamo alcune caratteristiche avanzate del TED. Grazie alle tecniche esposte ti sara' possibile ottenere nuovi tipi di effetti grafici e di animazione.

6.2 CARATTERI PROGRAMMABILI

Sai gia' come il COMMODORE 16 possa visualizzare i caratteri sul video, e sai anche che le immagini di questi caratteri (una serie di 8 byte per ogni carattere) sono custodite in ROM perche' non vengano perse ogni volta che spegni il calcolatore. Esiste pero' un modo per "dire" a TED di leggere le descrizioni dei caratteri in RAM e per indicargli l'indirizzo del primo byte del primo carattere. Per far leggere TED dalla RAM devi porre a 0 il bit 2 del registro 12 (indirizzo 65298 ($FF12)) con l'istruzione:

```
POKE 65298, PEEK(65298) AND 251
```

Per indicare l'indirizzo del primo byte del primo carattere, devi porre nei bit 7-2 del registro 13 (indirizzo 65299 ($FF13)) la parte piu' significativa dell'indirizzo stesso, con l'istruzione

```
POKE 65299, (PEEK(65299) AND 3)+N
dove N*256 = indirizzo desiderato
e N e' un numero divisibile per 4
```

Proviamo a programmare un carattere: per esempio il simbolo CBM che e' disegnato sul tasto in basso a sinistra della tastiera. Usando la stessa tecnica con

cui sono definiti normalmente i caratteri, costruiamo una matrice 8 per 8 e riempiamola usando 0 per i punti spenti e 1 per quelli accesi:

```
0 0 0 1 1 0 0 0          * *
0 0 1 1 1 0 0 0        * * *
0 1 1 0 0 1 1 1      * *     * * *
0 1 1 0 0 1 1 0      * *     * *
0 1 1 0 0 1 1 1      * *     * * *
0 0 1 1 1 0 0 0        * * *
0 0 0 1 1 0 0 0          * *
0 0 0 0 0 0 0 0
```

Ora dobbiamo fare un po' di conti: la prima riga (cioe' il primo byte del carattere) contiene il numero binario 00011000, cioe' 24 decimale. Facendo lo stesso conto per le altre righe otteniamo i numeri decimali 24, 56, 103, 102, 103, 56, 24, 0. Questi numeri devono essere ora memorizzati nei primi 8 byte della nuova descrizione dei caratteri, programmando cosi' il carattere che ha D/CODE 0, cioe' la chiocciola (vedi Appendice D). Il programma GRAF1 sposta la descrizione dei caratteri all'indirizzo 14336 ($3800), programma il carattere di D/CODE 0 con il simbolo CBM e il carattere di D/CODE 32 (lo spazio) con uno spazio:

```
0 REM GRAF1
10 COLOR0,1:COLOR1,7,5:COLOR4,1
20 POKE65299,(PEEK(65299)AND3)+56
30 POKE65298,PEEK(65298)AND251
40 FORI=0TO7
50 READA:POKE14336+I,A:POKE14592+I,0
60 NEXTI
70 PRINTCHR$(147)CHR$(142)"@"
80 GETKEYA$
90 POKE65299,(PEEK(65299)AND3)+208
100 POKE65298,PEEK(65298)OR4
110 DATA24,56,103,102,103,56,24,0
```

Fai girare il programma: vedrai comparire sullo schermo, in alto a sinistra, il carattere che abbiamo programmato. Se premi un qualunque tasto (tranne STOP) il carattere tornera' ad essere una chiocciola e il programma finira'; se premi il tasto STOP il programma finisce senza riportare la descrizione dei caratteri in ROM: quindi invece di vedere i caratteri normali vedrai, premendo i tasti, dei caratteri casuali (tutti i caratteri di questo nuovo set che non abbiamo ancora programmato).

COMMENTO A GRAF1:

.10: schermo e bordo neri, caratteri azzurri.

.20: descrizione dei caratteri in 14336 ($3800).

.30: TED legge da RAM.

.40/60: programma il carattere di D/CODE 0 con i dati relativi al simbolo CBM, e il carattere di D/CODE 32 (indirizzo di partenza 14336+32*8 = 14592) con una serie di zeri per ottenere uno spazio.

.70: pulisce lo schermo, seleziona il set di caratteri maiuscolo, e visualizza il carattere di D/CODE 0.

.80: attende che sia premuto un tasto.

.90: descrizione dei caratteri in $D000.

.100: TED legge da ROM.

.110: dati relativi al simbolo CBM.

ATTENZIONE a non fare errori quando TED legge da RAM: infatti quando il sistema da' una segnalazione di errore mette automaticamente TED in condizioni di leggere la ROM e, se l'indirizzo della descrizione dei caratteri era stato cambiato, TED si trovera' a leggere dei dati da indirizzi di ROM non esistenti; il risultato sara' ovviamente spiacevole.

6.3 COPIA DEI CARATTERI DA ROM

Non sempre, quando definisci un nuovo set di caratteri, vuoi ottenere un risultato completamente diverso dal contenuto della ROM: potrebbe capitarti di voler avere a disposizione tutte le lettere del set del COMMODORE 16 e cambiare solo i caratteri grafici, o cambiare le lettere e conservare le cifre e i segni di punteggiatura. In questo caso dovrai copiare su RAM la parte del contenuto della ROM che ti interessa e poi definire, come hai gia' visto, gli altri caratteri. Purtroppo il BASIC del COMMODORE 16 quando legge un dato dalla memoria lo legge sempre dalla RAM. L'unico modo di leggere i dati dalla ROM dei caratteri e' quello di usare una semplice routine in linguaggio macchina. Il programma GRAF2 ha il compito di caricare in memoria (dall'indirizzo 8192 ($2000)) il programma GRAF3 e di farlo eseguire:

```
0 REM GRAF2
10 COLOR0,1:COLOR1,7,5:COLOR4,1
20 POKE56,56:POKE55,0:CLR:PRINTCHR$(142)
30 POKE65299,(PEEK(65299)AND3)+56
40 POKE65298,PEEK(65298)AND251
50 FORI=0TO31:READA:POKE8192+I,A:NEXT
```

```
60 SYS8192:NEW
1000 DATA169,208,133,4,169,56,133,6
1010 DATA160,0,132,3,132,5,177,3
1020 DATA145,5,200,208,249,230,4,230
1030 DATA6,165,4,201,216,208,239,96
```

```
MONITOR

   PC  SR AC XR YR SP
; 0000 00 00 00 00 F8

. 2000  A9 D0     LDA #$D0
. 2002  85 04     STA $04
. 2004  A9 38     LDA #$38
. 2006  85 06     STA $06
. 2008  A0 00     LDY #$00
. 200A  84 03     STY $03
. 200C  84 05     STY $05
. 200E  B1 03     LDA ($03),Y
. 2010  91 05     STA ($05),Y
. 2012  C8        INY
. 2013  D0 F9     BNE $200E
. 2015  E6 04     INC $04
. 2017  E6 06     INC $06
. 2019  A5 04     LDA $04
. 201B  C9 D8     CMP #$D8
. 201D  D0 EF     BNE $200E
. 201F  60        RTS
```

Potrebbe sembrare che il programma GRAF2 non produca alcun effetto; in realta' i dati relativi ai caratteri si trovano ora in RAM. Prova infatti a scrivere una chiocciola sullo schermo e a dare quindi le istruzioni

FOR I=0 TO 7: POKE 14336+I,0: NEXT I.

Appena premi RETURN la chiocciola sparisce perche' hai cancellato in RAM i dati relativi alla chiocciola del set maiuscolo: se selezioni il set minuscolo le chiocciole riappariranno la' dove erano sparite.

COMMENTO A GRAF2

.10: schermo e bordo neri, caratteri azzurri.

.20: aggiorna i puntatori di fine memoria a 14336 ($3800) in modo che le variabili non cancellino i caratteri programmati. Seleziona il set di caratteri maiuscoli.

.30: descrizione dei caratteri in 14336 ($3800).

.40: TED legge da RAM.

.50: pone il programma GRAF3 in memoria

.60: esegue la routine GRAF3 e cancella il programma.

.1000/1030: contengono i dati relativi a GRAF3.

COMMENTO A GRAF3
(con riferimento agli indirizzi esadecimali)

.2000/2002 pone 208 ($D0) nel byte piu' significativo del puntatore 03-04.

.2004/2006 pone 56 ($38) nel byte piu' significativo del puntatore 05-06.

.2008/200C pone 0 ($00) nei byte meno significativi dei due puntatori: 03-04 punta cosi' all'inizio della ROM dei caratteri e 05-06 punta all'inizio dell'area RAM in cui vogliamo trasferire i caratteri.

.200E/2013 trasferisce una pagina ($100 corrisponde a 256 byte) dal byte puntato da 03-04 al byte puntato da 05-06.

.2015/2017 incrementa i piu' significativi dei due puntatori in modo che puntino alla prossima pagina.

.2019/201D se non ha ancora copiato 8 pagine (da $D000 a $D7FF) continua.

.201F torna al BASIC.

Il prossimo programma, GRAF4, ha il compito di trasferire in RAM i caratteri e sostituire le lettere del set MAIUSCOLO/GRAFICO con delle lettere gotiche. Esso usa ancora la routine GRAF3.

```
0 REM GRAF4
10 COLOR0,2,7:COLOR1,1:COLOR4,7,6
20 PRINTCHR$(142)
30 POKE65299,(PEEK(65299)AND3)+56
40 POKE65298,PEEK(65298)AND251
50 FORI=0TO31:READA:POKE8192+I,A:NEXT
60 SYS8192
70 FORI=0TO215:READA:POKE14336+I,A:NEXT
80 SYS32768
1000 DATA169,208,133,4,169,56,133,6
1010 DATA160,0,132,3,132,5,177,3
1020 DATA145,5,200,208,249,230,4,230
1030 DATA6,165,4,201,216,208,239,96
1040 DATA60,102,110,110,96,98,60,0
1050 DATA48,72,20,34,62,34,65,0
1060 DATA92,34,66,124,66,34,92,0
1070 DATA28,34,84,80,80,34,28,0
1080 DATA88,100,66,66,66,100,88,0
1090 DATA92,34,64,112,64,34,92,0
1100 DATA92,34,32,120,32,32,64,0
1110 DATA28,34,64,94,98,62,2,6
1120 DATA28,34,32,60,34,34,36,0
1130 DATA2,60,72,8,10,60,64,0
1140 DATA1,2,2,2,34,68,56,0
1150 DATA66,36,40,112,40,36,66,0
1160 DATA24,36,32,32,32,33,94,0
1170 DATA84,42,42,106,42,42,64,0
1180 DATA66,50,42,106,42,42,68,0
1190 DATA28,34,81,81,81,34,28,0
1200 DATA92,34,34,124,32,32,64,0
1210 DATA56,84,34,2,12,26,124,0
1220 DATA92,34,34,120,36,34,66,0
```

```
1230 DATA2,60,64,60,2,60,64,0
1240 DATA1,126,48,80,80,33,30,0
1250 DATA33,82,18,18,18,18,12,0
1260 DATA76,178,34,34,34,20,8,0
1270 DATA128,92,82,82,82,84,40,0
1280 DATA34,84,12,8,24,37,66,0
1290 DATA66,164,36,36,26,66,60,0
1300 DATA62,2,4,8,16,32,62,0
```

COMMENTO A GRAF4

.10 ripristina le condizioni iniziali dello schermo.
.20 seleziona il set MAIUSCOLO/GRAFICO.
.30/40 indica a TED la nuova descrizione dei caratteri.
.50 carica in memoria GRAF3.
.60 esegue GRAF3.
.70 pone nella memoria dei caratteri i dati riguardanti i caratteri gotici.
.80 inizializza il BASIC.
.1000/1030 dati relativi a GRAF3.
.1040/1300 dati relativi ai nuovi caratteri.

Dopo aver fatto girare questo programma il COMMODORE 16 e' pronto a funzionare ma puo' rovinare la descrizione dei caratteri con le variabili stringa. Per evitare cio' puoi dare le istruzioni
POKE 56,56: POKE 55,0: CLR
Queste istruzioni abbassano la fine della memoria a 14336 ($3800), dove inizia la descrizione dei caratteri.
Ricorda sempre che un errore in queste condizioni porta TED a leggere da una ROM inesistente. Se ti dovesse capitare di avere una segnalazione di errore e non vuoi resettare puoi dare l'istruzione
POKE 65298,PEEK(65298)AND251
anche se non vedi i caratteri che stai scrivendo; essa resetta il bit di posizione 2 al valore 0.

6.4 PROGRAMMI PER LA CREAZIONE DEI CARATTERI

Programmare nuovi caratteri con il COMMODORE 16 e' facile ma non molto immediato: e' difficile cioe'

"vedere" che 24, 56, 103, 102, 103, 56, 24, 0 rappresentano un simbolo grafico. Sarebbe molto piu' comodo se si potesse disegnare direttamente il carattere sullo schermo e farlo memorizzare senza tanti calcoli, PEEK e POKE. Il programma GRAF5 fa precisamente questo. E' commentato come al solito e puoi quindi eliminare facilmente certe parti che non ti interessano o aggiungerne altre per renderlo piu' adatto ai tuoi scopi.

```
0 REM GRAF5
10 COLOR0,1:COLOR1,7,5:COLOR4,1
15 PRINTCHR$(142)CHR$(8)
20 POKE56,56:POKE55,0:CLR
30 US$=CHR$(147)+"              PREMI F1 PER USCIRE"
40 KEY1,CHR$(133)
100 PRINTCHR$(147)
110 PRINT"OPZIONI  :":PRINT:PRINT
120 PRINT"1 CREA CARATTERE":PRINT
130 PRINT"2 MEMORIZZA CARATTERE":PRINT
140 PRINT"3 CORREGGE CARATTERE":PRINT
150 PRINT"4 MOSTRA CARATTERI":PRINT
160 PRINT"5 SALVA SET DI CARATTERI":PRINT
170 PRINT"6 CARICA SET DI CARATTERI":PRINT
180 PRINT"7 FINE"
190 GETA$:IFVAL(A$)=0THEN190
195 I=VAL(A$)
200 ONIGOSUB1000,2000,3000,4000,5000,6000,7000
210 GOTO100
1000 PRINTUS$
1010 FORI=9TO16:CHAR1,16,I,"++++++++":NEXT
1060 PC=2424:XC=0:YC=0
1070 POKEPC,PEEK(PC)+128
1080 GETKEYA$
1090 A=ASC(A$):SP=0
1100 IFA=29ANDXC<7THENXC=XC+1:PC=PC+1:SP=1
1110 IFA=157ANDXC>0THENXC=XC-1:PC=PC-1:SP=-1
1120 IFA=17ANDYC<7THENYC=YC+1:PC=PC+40:SP=40
1130 IFA=145ANDYC>0THENYC=YC-1:PC=PC-40:SP=-40
1140 IFA=32THENPOKEPC+1024,81
1150 IFA=20THENPOKEPC+1024,43
1160 POKEPC-SP,PEEK(PC-SP)-128
1170 IFA<>133THEN1070
1180 FORI=0TO7
1190 BY(I)=0
1200 FORJ=0TO7
1210 BY(I)=BY(I)-2↑J*(PEEK(3448+7-J+40*I)=81)
1220 NEXTJ:NEXTI
1230 RETURN
2000 PRINTCHR$(147)
2010 PRINT"1 NORMALE":PRINT
2020 PRINT"2 REVERSATO":PRINT
2030 PRINT"3 SIMMETRIA VERTICALE":PRINT
2040 PRINT"4 SIMMETRIA ORIZZONTALE":PRINT
2050 PRINT"5 RUOTATO DI 90 GRADI  ";
2055 PRINT"IN SENSO ORARIO":PRINT
2060 GETKEYA$
```

```
2070 A=VAL(A$):IFA=0ORA>5THEN2060
2080 INPUT"CARATTERE NUMERO ";NC%
2090 IFNC%>255ORN%<0THEN2080
2100 ONAGOSUB2200,2300,2400,2500,2600
2110 FORI=0TO7:POKE14336+8*NC%+I,BY(I):NEXT
2120 RETURN
2200 RETURN
2300 FORI=0TO7:BY(I)=255-BY(I):NEXT
2310 RETURN
2400 FORI=0TO7
2410 FORJ=0TO3
2420 BD=(BY(I)AND2↑J)/2↑J:BY(I)=BY(I)-2↑J*BD
2430 BS=(BY(I)AND2↑(7-J))/2↑(7-J)
2435 BY(I)=BY(I)-2↑(7-J)*BS
2440 BY(I)=BY(I)+2↑J*BS:BY(I)=BY(I)+2↑(7-J)*BD
2450 NEXTJ,I
2460 RETURN
2500 FORI=0TO3
2510 T=BY(I)
2520 BY(I)=BY(7-I)
2530 BY(7-I)=T
2540 NEXTI
2550 RETURN
2600 FORI=0TO7
2610 B2(I)=BY(I):BY(I)=0
2620 NEXTI
2630 FORI=0TO7
2640 FORJ=0TO7
2645 EX=(B2(I)AND2↑(7-J))/2↑(7-J)
2650 BY(J)=BY(J)OR(2↑(EX*I)*-(EX<>0))
2660 NEXTJ,I
2670 RETURN.
3000 PRINTCHR$(147)
3010 INPUT"CARATTERE DA CORREGGERE ";NC%
3020 IFNC%>255ORN%<0THEN3010
3030 PRINTUS$
3040 FORI=1TO8:PRINTCHR$(17);:NEXTI
3050 FORI=0TO7
3060 FORJ=1TO16:PRINTCHR$(32);:NEXT
3070 FORJ=0TO7
3080 IFPEEK(14336+8*NC%+I)AND2↑(7-J)THENSI=-1
3085 IFSITHENPRINTCHR$(209);:SI=0:GOTO3100
3090 PRINTCHR$(43);
3100 NEXTJ:PRINT:NEXTI
3110 GOTO1060
4000 PRINTCHR$(147)"PREMI F1 PER TORNARE"
4005 PRINTCHR$(17)"PREMI I TASTI CORRISPONDENTI"
4006 PRINT"AI CARATTERI CHE VUOI VEDERE"
4010 FORI=1TO2000:NEXTI
4020 PRINTCHR$(147);
4030 POKE65298,PEEK(65298)AND251
4035 POKE65299,(PEEK(65299)AND3)+56
4040 GETKEYA$
4050 IFA$<>CHR$(133)THENPRINTA$;:GOTO4040
4060 POKE65298,PEEK(65298)OR4
4070 POKE55299,(PEEK(65299)AND3)+208
4080 COLOR1,7,5:RETURN
5000 PRINTL4R$(147)
5010 PRINT"LISCO O NASTRO ?"
5015 GETDV$:iFDV$<>"N"ANDDV$<>"D"THEN5015
5020 PRINT
5030 INPUT"SET NUMERO ";NS%
5040 PRINT
```

```
5050 PRINT"FINO A CHE CARATTERE VUOI SALVARE ";
5055 INPUTNC%
5060 IFNC%>255THEN5040
5070 IFDV$="N"THEN5500
5080 OPEN1,8,2,"@:SET #"+STR$(NS%)+",S,W"
5082 PRINT:PRINTDS$
5085 IFDSTHENFORI=1TO2000:NEXT:CLOSE1:RETURN
5090 PRINT#1,CHR$(NC%);:FORI=0TO(NC%+1)*8-1
5100 PRINT#1,CHR$(PEEK(14336+I));
5110 NEXTI
5120 CLOSE1
5130 RETURN
5500 OPEN1,1,1,"SET #"+STR$(NS%)
5510 PRINT#1,CHR$(NC%);:FORI=0TO(NC%+1)*8-1
5520 PRINT#1,CHR$(PEEK(14336+I));
5530 NEXTI
5540 CLOSE1
5550 RETURN
6000 PRINTCHR$(147)
6010 PRINT"DISCO O NASTRO ?"
6015 GETDV$:IFDV$<>"N"ANDDV$<>"D"THEN6015
6020 PRINT
6030 INPUT"SET NUMERO ";NS%
6070 IFDV$="N"THEN6500
6080 OPEN1,8,2,"SET #"+STR$(NS%)+",S,R"
6082 PRINT:PRINTDS$
6085 IFDSTHENFORI=1TO2000:NEXT:CLOSE1:RETURN
6090 GET#1,A$:NC%=ASC(A$):FORI=0TO(NC%+1)*8-1
6100 GET#1,A$:POKE14336+I,ASC(A$+CHR$(0))
6110 NEXTI
6120 CLOSE1
6130 RETURN
6500 OPEN1,1,0,"SET #"+STR$(NS%)
6510 GET#1,A$:NC%=ASC(A$):FORI=0TO(NC%+1)%*8-1
6520 GET#1,A$:POKE14336+I,ASC(A$+CHR$(0))
6530 NEXTI
6540 CLOSE1
6550 RETURN
7000 PRINTCHR$(17)"SICURO ?"
7010 GETKEYA$
7020 IFA$="S"THENSYS32768
7030 RETURN
```

Il programma GRAF5 puo' essere diviso in 14 sezioni ben distinte:

-1: 10-40: inizializza il calcolatore.

-2: 100-210: mostra il menu' e salta all'opzione richiesta.

-3: 1000-1230: ti permette di disegnare un nuovo carattere e riempie un vettore con i dati relativi al carattere disegnato.

-4: 2000-2120: mostra il menu' di memorizzazione dei caratteri, salta alla routine per il tipo di operazione richiesta sul vettore e memorizza il carattere.

-5: 2200: lascia il vettore invariato: e' stata messa per rispondere alla chiamata della linea 2100.

-6: 2300-2310: pone nel vettore il "negativo" del carattere contenuto prima.

-7: 2400-2460: pone nel vettore il carattere che gode di simmetria verticale rispetto a quello contenuto prima.

-8: 2500-2550: pone nel vettore il carattere che gode di simmetria orizzontale rispetto a quello contenuto prima.

-9: 2600-2670: pone nel vettore un carattere ruotato di 90 gradi in senso orario rispetto a quello contenuto prima. Chiamando 2 volte questa routine otterrai una rotazione di 180 gradi. Chiamandola 3 volte otterrai una rotazione di 90 gradi in senso antiorario.

-10: 3000-3110: riempie il vettore con i dati relativi al carattere che vuoi correggere e, saltando alla linea 1060, ti permette la correzione.

-11: 4000-4080: ti permette di vedere i caratteri che hai creato.

-12: 5000-5550: salva su disco o su nastro i caratteri che hai programmato.

-13: 6000-6550: carica da disco o da nastro un set salvato precedentemente.

-14: 7000-7030: chiude il programma.

COMMENTO A GRAF5 SEZIONE PER SEZIONE

SEZIONE 1

.10: schermo e sfondo neri, caratteri azzurri.

.15: set maiuscolo, disabilita la funzione di SHIFT-CBM.

.20: fine della memoria a 14336 ($3800).

.30: inizializza la costante US$.

.40: assegna a F1 il CHR$(133) per poterlo usare con l'istruzione GETKEY. Il tasto F1 serve per uscire dalle fasi del programma e tornare al menu' principale.

SEZIONE 2

.100/180: mostra le opzioni.

.190: accetta un tasto tra 1 e 9.

.195: pone nella variabile I il valore del tasto premuto.

.200: se I<7 salta alla routine richiesta.
.210: torna a mostrare le opzioni.

SEZIONE 3:

E' la sezione piu' complicata del programma: per comprenderne bene il funzionamento possiamo dividerla in 3 sottosezioni.

.A) . 1000/1010: disegna una griglia 8X8 su cui creare il carattere.

.B) . 1060/1170: permette di disegnare il carattere.

.C ; 1180-1230: riempie il vettore BY(7) con i dati relativi al carattere disegnato sulla griglia.

Torniamo al commento linea per linea:

.1000: pulisce lo schermo e scrive in alto PREMI F1 PER USCIRE.

.1010: disegna la griglia.

.1060: pone nella variabile PC (posizione cursore) l'indirizzo del byte della mappa degli attributi corrispondente al "+" della griglia in alto a sinistra. Azzera le variabili XC e YC (coordinate X e Y del cursore).

.1070: pone a 1 il bit 7 del byte della mappa degli attributi corrispondente al cursore. In questo modo il carattere che corrisponde al cursore, lampeggia (vedi Paragrafo 4.8).

.1080: attende la pressione di un tasto.

.1090: A = codice ASCII ricevuto da tastiera e SP=0 (SP=spostamento che verra' effettuato).

.1100: se il carattere ricevuto e' "CURSORE A DESTRA" e il cursore non e' all'estrema destra della griglia (XC=7) allora incrementa XC e PC e pone SP=1.

.1110: se il carattere ricevuto e' "CURSORE A SINISTRA" e il cursore non e' all'estrema sinistra della griglia (XC=0) allora decrementa XC e PC e pone SP=-1.

.1120: se il carattere ricevuto e' "CURSORE IN BASSO" e il cursore non e' sull'ultima linea della griglia (YC=7) allora incrementa YC, somma 40 a PC e pone SP=40 (40 e' il numero di caratteri contenuti in una linea dello schermo).

.1130: se il carattere ricevuto e' "CURSORE IN ALTO" e il cursore non e' sulla prima linea della griglia (YC=0) allora decrementa YC, sottrae 40 a PC e pone SP=-40.

.1140: se il carattere ricevuto da tastiera e' SPAZIO pone nella posizione della mappa video corrispondente alla posizione del cursore una pallina per indicare che quel punto e' "acceso" (1024 e' la differenza tra l'indirizzi della mappa degli attributi e i corrispondenti indirizzi della mappa video).
.1150: se il carattere ricevuto da tastiera e' DELETE pone nella posizione della mappa video corrispondente alla posizione del cursore un "+" per indicare che quel punto e' "spento".
.1160: pone a 0 il bit 7 del byte della mappa degli attributi corrispondente alla vecchia posizione del cursore (PC-SP).
.1170: se non e' stato premuto F1 torna alla linea 1070 dove fa lampeggiare la posizione corrente del cursore.
.1180: per ogni riga della griglia esegue fino a 1220.
.1190: pone a 0 il corrispondente elemento del vettore.
.1200: per ogni elemento di una riga esegue fino a 1220
.1210: somma alla variabile corrispondente alla riga interessata 2 elevato alla posizione dell'elemento di riga considerato (partendo da destra), se in tale posizione vi e' una pallina; altrimenti somma 0.
.1220: chiude il ciclo degli elementi e quello delle righe. A questo punto nel vettore BY sono contenuti gli 8 numeri necessari per programmare il carattere disegnato sulla griglia.
.1230: fine della routine.

SEZIONE 4

.2000/2055: mostra le opzioni relative alla memorizzazione dei caratteri.
.2060/2070: accetta dalla tastiera un numero tra 1 e 5 e lo pone nella variabile A.
.2080/2090: accetta un numero tra 0 e 255 e lo pone nella variabile NC% (il carattere che si vuole programmare).
.2100: salta alla routine di modifica del vettore richiesta.
.2110: memorizza il carattere ponendo in 8 byte di memoria a partire da 14336+8*NC% il contenuto del vettore.
.2120: torna dalla routine.

SEZIONE 5

.2200: e' stata messa per rispondere alla chiamata della linea 2100.

SEZIONE 6

.2300: pone in ogni elemento del vettore la differenza tra 255 e il valore contenuto prima. Questa operazione compiuta su numeri minori di 256 (come nel nostro caso), nega gli 8 bit del numero.
.2310: torna dalla routine.

SEZIONE 7

.2400: per ogni elemento del vettore esegue fino a 2450.
2410: per J da 0 a 3 esegue fino a 2450.
.2420: pone il valore del bit j-esimo dell'elemento considerato del vettore nella variabile BD (bit di destra) e lo azzera.
.2430/2435: pone il valore del bit (7-j)-esimo dell'elemento considerato del vettore nella variabile BS (bit di sinistra) e lo azzera.
.2440: pone il bit di sinistra al posto del bit di destra e viceversa.
.2450: chiude i due cicli.
.2460: torna dalla routine.

SEZIONE 8

.2500: per i primi 4 elementi del vettore esegue fino a 2540.
.2510: pone nella variabile T (temporanea) il valore dell'elemento considerato.
.2520: pone nell'elemento considerato il valore dell'elemento simmetrico del vettore.
.2530: pone nell'elemento simmetrico il valore di T.
.2540: chiude il ciclo.
.2550: torna dalla routine.

SEZIONE 9

.2600/2620: trasferisce il vettore BY nel vettore B2 e lo azzera.
.2630: per ogni bit degli elementi del vettore esegue fino a 2660 (indice I).

.2640: per ogni elemento del vettore esegue fino a 2660 (indice J).
.2645/2650: pone il bit considerato dell'elemento in esame uguale al bit j¬esimo dell'elemento simmetrico all'i¬esimo.
.2660: chiude i due cicli.
.2670: torna dalla routine.

SEZIONE 10
.3000/3020: chiede il numero (D/CODE) del carattere da correggere. Accetta un numero tra 0 e 255 e lo pone nella variabile NC%.
.3030: pulisce lo schermo e scrive centrato in alto PREMI F1 PER USCIRE.
.3040: sposta il cursore piu' in basso di 4 linee.
.3050: per 8 volte esegue fino a 3100.
.3060: sposta il cursore a destra di 16 posizioni.
.3070/3100: scrive un "+" se il bit considerato contiene 0, una pallina altrimenti.
.3110: salta alla routine di programmazione del carattere.

SEZIONE 11
.4000/4006: fornisce le istruzioni all'utente.
.4010: attende circa 10 secondi.
.4020: cancella lo schermo.
.4030/4035: descrizione dei caratteri in RAM a partire da 14336 ($3800).
.4040: attende la pressione di un tasto.
.4050: se il tasto premuto non e' F1, scrive il carattere (che puo' essere anche un carattere di controllo del cursore o del colore) e torna ad attendere il prossimo.
.4060/4080: se e' stato premuto F1 riassume la ROM come descrizione dei caratteri, passa a scritte blu' e esce dalla routine.

SEZIONE 12
.5000: pulisce lo schermo.
.5010/5015: pone in DV$ la periferica scelta.
.5020/5030: pone in NS% il numero distintivo del set di caratteri.
.5040/5060: pone in NC% (D/CODE dell'ultimo carattere che si vuole salvare) un numero tra 0 e 255.
.5070: se la periferica scelta e' il nastro salta a 5500.
.5080/5085: apre il file su disco e torna se si verifica un errore.

.5090/5110: scrive su disco il numero di caratteri da salvare seguito dai dati relativi ai caratteri.
.5120/5130: chiude il file e torna dalla routine.
.5500: apre il file su nastro.
.5510/5530: scrive su nastro il numero di caratteri da salvare seguito dai dati relativi ai caratteri.
.5540/5550: chiude il file e torna dalla routine.

SEZIONE 13

.6000: pulisce lo schermo.
.6010/6015: pone in DV$ la periferica scelta.
.6020/6030: pone in NS% il numero distintivo del set di caratteri.
.6070: se la periferica scelta e' il nastro salta a 6500.
.6080/6085: apre il file su disco e torna se si verifica un errore.
.6090/6110: legge da disco il numero di caratteri da caricare e quindi carica i dati relativi ai caratteri.
.6120/6130: chiude il file e torna dalla routine.
.6500: apre il file su nastro.
.6510/6530: legge da nastro il numero di caratteri da caricare e quindi carica i dati relativi ai caratteri.
.6540/6550: chiude il file e torna dalla routine.

SEZIONE 14

.7000: porta il cursore piu' giu' di una linea e ti chiede se sei sicuro di voler abbandonare il programma.
.7010: attende la pressione di un tasto.
.7020: se la risposta e' "S" allora salta alla routine di inizializzazione del BASIC.
.7030: altrimenti torna dalla routine.

Con questo programma e' quindi possibile creare in memoria nuovi caratteri e salvarli su disco o cassetta. Se, in un programma che usa caratteri definiti, vuoi evitare di caricare i dati relativi ai caratteri da nastro, ma preferisci che siano gia' nel programma sotto forma di linee DATA, puoi usare il programma GRAF6, che converte i dati contenuti dal byte 14336 ($3800) in poi in linee DATA.

```
10 REM GRAF6
20 INPUT"FINO A CHE CARATTERE ";C%
50 PRINTCHR$(147);
60 N=1000+I*10:GOSUB160:PRINTN$"DATA";
70 FORJ=0TO7
80 N=PEEK(14336+I*8+J):GOSUB160:PRINTN$",";
90 NEXT
100 PRINTCHR$(20)
110 PRINT"I="I+1":C%="C%;
120 IFI<C%THENPRINT":GOTO50":GOTO140
130 PRINT":GOTO170"
140 POKE239,3:POKE1319,19:POKE1320,13
150 POKE1321,13:END
160 N$=STR$(N):N$=RIGHT$(N$,LEN(N$)-1):RETURN
170 POKE239,3:POKE1319,19:POKE1320,13:POKE1321,13:
    K=K+10:PRINTCHR$(147)K:PRINT"K="K":
    IFK<170THEN170"
```

GRAF6 puo' essere usato anche per convertire in linee DATA programmi in linguaggio macchina o altri tipi di dati presenti in memoria. Il programma si basa sul fatto che il COMMODORE 16, appena esce da un programma, scrive i caratteri che si trovano nel buffer della tastiera (da 1319 a 1328 ($0527/$0530)) e si comporta esattamente come se fossero stati premuti dall'utente.

COMMENTO A GRAF6

.20: chiede fino a che carattere vuoi convertire in linee DATA. Il programma convertira' 8X(C%+1) byte a partire dall'indirizzo 14336 ($3800).

.50: pulisce lo schermo.

.60: il numero N che deve essere assegnato alla prossima linea DATA e' 1000+I*10, trasforma il numero N in una stringa (appoggiandosi ad una routine in 160), scrive una stringa che contiene N e la parola BASIC DATA. Puoi scegliere da quale linea partire, sostituendo a 1000 il numero di linea che preferisci, e il passo con cui incrementare il numero di linea, sostituendo a 10 il passo desiderato.

.70/90: per gli 8 byte di ogni carattere pone in N il contenuto del byte interessato, lo trasforma in una stringa utilizzando la stessa routine di prima, scrive la stringa e una virgola.

.100: cancella l'ultima virgola.

.110: scrive sul video le istruzioni BASIC che incrementeranno l'indice I e riporranno in memoria il valore di C% (le variabili vengono infatti perse ogni volta che si introduce una nuova linea di programma).

.120: se il carattere trattato non e' l'ultimo, la linea 120 scrive GOTO 50 e salta a 140.

.130: se il carattere trattato e' l'ultimo, scrive GOTO 170.

.140/150: pone 3 nel byte 239 (il byte 239 indica il numero di caratteri validi nel buffer di tastiera), pone nel buffer di tastiera il codice ASCII di HOME e due volte il codice ASCII di RETURN e esce dal programma. A questo punto il COMMODORE 16 trova nel buffer di tastiera HOME e quindi porta il cursore sulla linea DATA, un RETURN e quindi introduce nel programma la linea DATA portando il cursore sulla seconda linea scritta dal programma; trova il secondo RETURN ed esegue quindi i comandi scritti sulla linea, aggiorna cioe' le variabili e salta a 50 o a 170.

.160: e' la routine che trasforma il numero N nella corrispondente stringa N$ usando la funzione STR$ e scartando il carattere che contiene il segno.

.170: usando il metodo appena descritto, cancella le linee da 10 a 170.

6.5 CARATTERI A SFONDO PROGRAMMABILE

Tu sai che quando scrivi un carattere con il COMMODORE 16 puoi sceglierne il colore, selezionando il colore 1, usando i codici di controllo del colore del cursore, o scrivendo nella mappa degli attributi il codice desiderato. In questo modo selezioni il colore dello "inchiostro" con cui il COMMODORE 16 scrive il carattere, ma quello della "carta" (lo sfondo) rimane sempre lo stesso. TED ha la capacita' di leggere 4 diversi colori di sfondo quando e' posto in modo SFONDO PROGRAMMABILE. In questo modo, infatti, quando TED legge dalla mappa video il codice del carattere da visualizzare, considera i 6 bit meno significativi come codice del carattere, e i due bit rimanenti come codice del colore dello sfondo. Hai cosi' a disposizione un set di soli 64 (2 elevato a 6) caratteri, ma puoi scegliere per ciascuno di essi un colore di sfondo su 4 possibili. Puoi naturalmente scegliere questi 4 colori tra i 121 a disposizione. Con il modo a sfondo programmabile puoi ottenere delle bellissime maschere o dei coloratissimi quadri grafici. I caratteri che si perdono con il modo a sfondo programmabile sono:

. tutto il set MINUSCOLO/MAIUSCOLO (TED non considera, infatti, il bit 2 del registro 13, quando e' in

modo SFONDO PROGRAMMABILE, non e' quindi possibile selezionare il set MINUSCOLO/MAIUSCOLO).
. tutti i caratteri in campo inverso.
. tutti i caratteri grafici.
Ti rimangono quindi a disposizione solo le lettere maiuscole, le dieci cifre, i segni di punteggiatura e delle operazioni matematiche.
Dei 4 colori-sfondo per i caratteri, uno e' quello dello schermo, i codici degli altri tre vanno posti nei registri del TED che rispondono agli indirizzi 65302, 65303, 65304 ($FF16, $FF17, $FF18). Se L e' il numero della luminosita' (tra 0 e 7) e C e' il numero del colore (tra 1 e 16), devi porre nei registri il numero L*16+C-1.
. il colore dello schermo viene dato come sfondo ai caratteri che hanno nei due bit piu' significativi del codice 00.
. il colore del registro di indirizzo 65302 (colore di sfondo 1) viene dato come sfondo ai caratteri che hanno nei due bit piu' significativi del codice 01.
. il colore del registro di indirizzo 65303 (colore di sfondo 2) viene dato come sfondo ai caratteri che hanno nei due bit piu' significativi del codice 10.
. il colore del registro di indirizzo 65304 (colore di sfondo 3) viene dato come sfondo ai caratteri che hanno nei due bit piu' significativi del codice 11.
Ricorda che, normalmente, il bit 7 vale 1 per i caratteri in campo inverso: nel modo a sfondo programmabile, quindi, se premi una A ottieni una A su sfondo normale, mentre RVS ON + A produce una A con colore di sfondo 2. Per le lettere e lo spazio vale anche la regola che SHIFT-LETTERA visualizza la lettera con colore di sfondo 1 e RVS ON + SHIFT-LETTERA visualizza la lettera con colore di sfondo 3. Questa regola non vale per i numeri e i rimanenti simboli. Alla fine del paragrafo riportiamo la Tabella 6.1; essa ti aiuta a ottenere facilmente i 64 simboli con i vari colori di sfondo. Per entrare in modo SFONDO PROGRAMMABILE basta porre a 1 il bit 6 del registro 6 del TED che risponde all'indirizzo 65286 ($FF06). L'istruzione da dare e' quindi la seguente:

```
POKE 65286, PEEK(65286) OR 64
```

mentre l'istruzione da dare per tornare al modo modo normale e':

```
POKE 65286, PEEK(65286) AND 191
```

Come prova di programmazione dello sfondo dei caratteri segue il programma GRAF7.

```
0 rem graf7
10 color0,1:color4,1
20 poke65286,peek(65286)or64
30 poke65302,92
40 poke65303,94
50 poke65304,82
60 printchr$(147)
70 color1,8,5
80 char1,13,6,"commodore 16"
90 color1,6,3
100 char1,9,9,"                    "
110 char1,9,10," PROGRAMMA DI PROVA "
120 char1,9,11,"                    "
130 color1,7,3
140 char1,13,13,chr$(18)+"           "
150 char1,13,14,chr$(18)+" caratteri "
160 char1,13,15,chr$(18)+"           "
170 color1,3,3
180 char1,7,17,chr$(18)+"                          "
190 char1,7,18,chr$(18)+" A SFONDO PROGRAMMABILE "
200 char1,7,19,chr$(18)+"                          "
210 getkeya$
220 poke65286,peek(65286)and191
```

COMMENTO A GRAF7

.10: schermo e bordo neri.

.20: entra in modo sfondo programmabile.

.30: colore di sfondo 1: colore 13 con luminosita' 5 (5*16+13-1=92).

.40: colore di sfondo 1: colore 15 con luminosita' 5 (5*16+15-1=94).

.30: colore di sfondo 1: colore 3 con luminosita' 5 (5*16+3-1=82).

.60: pulisce lo schermo.

.70: colore delle scritte giallo.

.80: scrive in colonna 13, riga 6, "COMMODORE 16".

.90: colore delle scritte verde.

.100: scrive in colonna 9, riga 9, 20 spazi shiftati.

.110: scrive in colonna 9, riga 10, la scritta shiftata " PROGRAMMA DI PROVA ".

.120: scrive in colonna 9, riga 11, 20 spazi shiftati.

.130: colore delle scritte blu.

.140: scrive in colonna 13, riga 13, 11 spazi reversati.

.150: scrive in colonna 13, riga 14, la scritta reversata " CARATTERI ".

.160: scrive in colonna 13, riga 15, 11 spazi reversati.
.170: colore delle scritte rosso.
.180: scrive in colonna 7, riga 17, 24 spazi shiftati e reversati.
.190: scrive in colonna 7, riga 18, la scritta shiftata e reversata " A SFONDO PROGRAMMABILE ".
.200: scrive in colonna 13, riga 15, 24 spazi shiftati e reversati.
.210: attende la pressione di un tasto.
.220: torna al modo normale.

Nota che in modo sfondo programmabile, il bit 7 dell'attributo viene ignorato, non e' possibile quindi ottenere caratteri lampeggianti (effetto FLASH). Ti proponiamo ora il programma GRAF8, leggermente piu' complicato, che usa il modo sfondo programmabile con i caratteri definiti in RAM: in questo esempio gli stessi caratteri vengono "disegnati" su sfondi diversi.

```
0 REM GRAF8
10 POKE56,56:POKE55,0:CLR:DM=1024
20 FORI=14336TO14527
30 READA
40 POKEI,A
50 NEXT
60 FORI=0TO7:POKE14592+I,0:NEXT
70 PRINTCHR$(147)
80 COLOR0,14,6:COLOR4,2,5
90 POKE65302,85
100 POKE65303,73
110 POKE65304,103
120 POKE65299,(PEEK(65299)AND3)+56
121 POKE65298,PEEK(65298)AND251
130 POKE65286,PEEK(65286)OR64
140 COLOR1,3,3:CHAR1,39,7,"V"
150 FORI=0TO2
160 CHAR1,38-I,8+I,LEFT$("VWWW",I+2)
170 NEXT
180 FORI=3592TO3751:POKEI,96:NEXT
190 FORI=3752TO3911:POKEI,160:NEXT
200 FORI=3912TO4071:POKEI,96:NEXT
210 FORI=0TO2:FORJ=0TO2:POKE3549+J+I*40,224
220 NEXT:NEXT
230 OG=3588:T=1:GOSUB500
240 OG=3678:T=0:GOSUB500
250 OG=3538:T=0:GOSUB500
260 OG=3848:T=1:GOSUB500
270 GETKEYA$
280 POKE65286,PEEK(65286)AND191
290 POKE65299,(PEEK(65299)AND3)+208
300 POKE65298,PEEK(65298)OR4
310 PRINTCHR$(147):END
500 IFTTHEN620
```

```
510 POKEOG,(PEEK(OG)AND196)+0
511 POKEOG-DM,113:OG=OG+1
520 POKEOG,(PEEK(OG)AND196)+1
521 POKEOG-DM,113:OG=OG+39
530 POKEOG,(PEEK(OG)AND196)+2
531 POKEOG-DM,54:OG=OG+1
540 POKEOG,(PEEK(OG)AND196)+3
541 POKEOG-DM,54:OG=OG+38
550 POKEOG,(PEEK(OG)AND196)+6
551 POKEOG-DM,54:OG=OG+1
560 POKEOG,(PEEK(OG)AND196)+4
561 POKEOG-DM,54:OG=OG+1
570 POKEOG,(PEEK(OG)AND196)+5
571 POKEOG-DM,54:OG=OG+39
580 POKEOG,(PEEK(OG)AND196)+7
581 POKEOG-DM,113:OG=OG+1
590 POKEOG,(PEEK(OG)AND196)+8
591 POKEOG-DM,113:OG=OG+39
600 POKEOG,(PEEK(OG)AND196)+9
601 POKEOG-DM,113:OG=OG+1
610 POKEOG,(PEEK(OG)AND196)+10
611 POKEOG-DM,113:RETURN
620 POKEOG,(PEEK(OG)AND196)+12
621 POKEOG-DM,113:OG=OG+1
630 POKEOG,(PEEK(OG)AND196)+11
631 POKEOG-DM,113:OG=OG+39
640 POKEOG,(PEEK(OG)AND196)+14
641 POKEOG-DM,54:OG=OG+1
650 POKEOG,(PEEK(OG)AND196)+13
651 POKEOG-DM,54:OG=OG+39
660 POKEOG,(PEEK(OG)AND196)+16
661 POKEOG-DM,54:OG=OG+1
670 POKEOG,(PEEK(OG)AND196)+15
671 POKEOG-DM,54:OG=OG+1
680 POKEOG,(PEEK(OG)AND196)+17
681 POKEOG-DM,54:OG=OG+38
690 POKEOG,(PEEK(OG)AND196)+19
691 POKEOG-DM,113:OG=OG+1
700 POKEOG,(PEEK(OG)AND196)+18
701 POKEOG-DM,113:OG=OG+39
710 POKEOG,(PEEK(OG)AND196)+21
711 POKEOG-DM,113:OG=OG+1
720 POKEOG,(PEEK(OG)AND196)+20
721 POKEOG-DM,113:RETURN
730 DATA0,0,1,3,7,15,31,31
740 DATA0,96,240,240,224,192,192,224
750 DATA31,63,63,127,127,63,31,7
760 DATA224,54,191,255,246,240,224,0
770 DATA7,31,62,125,251,247,207,159
780 DATA0,176,216,220,238,239,231,195
790 DATA0,0,0,0,1,7,7,15
800 DATA31,31,15,23,27,29,30,60
810 DATA192,128,128,225,243,119,127,62
820 DATA126,255,126,0,0,0,0,0
830 DATA28,0,0,0,0,0,0,0
840 DATA0,0,128,192,224,240,248,248
850 DATA0,6,15,15,7,3,3,7
860 DATA248,252,252,254,254,252,248,224
870 DATA7,108,253,255,111,15,7,0
880 DATA224,248,124,190,223,239,243,249
890 DATA0,13,27,59,119,247,231,195
900 DATA0,0,0,0,128,224,224,240
910 DATA248,248,240,232,216,184,120,60
```

```
920 DATA3,1,1,135,207,238,254,124
930 DATA126,255,126,0,0,0,0,0
940 DATA56,0,0,0,0,0,0,0
950 DATA1,3,7,15,31,63,127,255
960 DATA255,255,255,255,255,255,255,255
```

GRAF8 memorizza i dati relativi a 22 caratteri necessari a disegnare dei puffi e, grazie a una subroutine, li pone in un qualsiasi punto di uno schermo variopinto.

COMMENTO A GRAF8

.10: pone la fine della memoria a 14336 ($3800) per proteggere i caratteri dalle variabili e pone nella variabile DM (Differenza Mappe) la differenza tra il primo indirizzo della mappa degli attributi e il primo indirizzo della mappa video.

.20/50: pone in memoria i dati delle immagini dei caratteri.

.60: azzera il carattere 32 (lo spazio).

.70: pulisce lo schermo.

.80: schermo blu chiaro e bordo grigio.

.90/110: selezionano i tre colori di sfondo:
1=verde, 2=marrone, 3=giallo.

.120: descrizione dei caratteri a partire da 14336 ($3800).

.130: entra in modo sfondo programmabile.

.140/170: disegna un triangolo rosso (che sara' il tetto di una casetta) con i caratteri programmati. Nota che non puoi sovrapporre dei puffi al tetto perche' questo e' stato ottenuto con dei caratteri e non con degli spazi colorati.

.180: disegna una striscia di spazi shiftati (colore di sfondo 1).

.190: disegna una striscia di spazi reversati (colore di sfondo 2).

.200: disegna un'altra striscia di spazi shiftati (colore di sfondo 1).

.210/220: disegna un quadrato di 3X3 spazi shiftati e reversati (colore di sfondo 3) che sara' la casetta.

.230: disegna un puffo di tipo 1 (che guarda a sinistra), il cui angolo in alto a sinistra verra' posto nel byte di memoria video 3588, usando la routine in 500.

.240: disegna un puffo di tipo 0 (che guarda a destra), il cui angolo in alto a sinistra verra' posto nel byte di memoria video 3678, usando la routine in 500.

.250: disegna un puffo di tipo 0 (che guarda a destra), il cui angolo in alto a sinistra verra' posto nel byte di memoria video 3538, usando la routine in 500.
.260: disegna un puffo di tipo 1 (che guarda a sinistra), il cui angolo in alto a sinistra verra' posto nel byte di memoria video 3848, usando la routine in 500.
.270: attende che sia premuto un tasto.
.280: esce dal modo a sfondo programmabile.
.290/300: descrizione dei caratteri in ROM.
.310: pulisce lo schermo ed esce.
.500: inizia la routine che disegna i puffi. Salta se T<>0 cioe' se il puffo e' di tipo 1.
.510/610: ogni linea disegna uno degli 11 caratteri che formano un puffo di tipo 0 e pone nel corrispondente byte della mappa degli attributi il colore adatto. Per disegnare un carattere in un byte facciamo la AND tra il contenuto del byte e 196 per ottenere il valore dei bit 7 e 6, sommiamo quindi il codice del carattere e poniamo il risultato nel byte. In questo modo lasciamo inalterato il colore dello sfondo.
.620/720: ogni linea disegna uno degli 11 caratteri che formano un puffo di tipo 1 e pone nel corrispondente byte della mappa degli attributi il colore adatto.
.730/960: dati relativi ai caratteri dei puffi e del tetto della casa.

TABELLA 6.1

SIMB.	TASTI DA PREMERE			
	SFONDO	COLORE 1	COLORE 2	COLORE 3
@	@	SHIFT+*	RVS+@	RVS+SHIFT+*
A	A	SHIFT+A	RVS+A	RVS+SHIFT+A
B	B	SHIFT+B	RVS+B	RVS+SHIFT+B
C	C	SHIFT+C	RVS+C	RVS+SHIFT+C
D	D	SHIFT+D	RVS+D	RVS+SHIFT+D
E	E	SHIFT+E	RVS+E	RVS+SHIFT+E
F	F	SHIFT+F	RVS+F	RVS+SHIFT+F
G	G	SHIFT+G	RVS+G	RVS+SHIFT+G

TABELLA 6.1				
SIMB.	TASTI DA PREMERE			
	SFONDO	COLORE 1	COLORE 2	COLORE 3
H	H	SHIFT+H	RVS+H	RVS+SHIFT+H
I	I	SHIFT+I	RVS+I	RVS+SHIFT+I
J	J	SHIFT+J	RVS+J	RVS+SHIFT+J
K	K	SHIFT+K	RVS+K	RVS+SHIFT+K
L	L	SHIFT+L	RVS+L	RVS+SHIFT+L
M	M	SHIFT+M	RVS+M	RVS+SHIFT+M
N	N	SHIFT+N	RVS+N	RVS+SHIFT+N
O	O	SHIFT+O	RVS+O	RVS+SHIFT+O
P	P	SHIFT+P	RVS+P	RVS+SHIFT+P
Q	Q	SHIFT+Q	RVS+Q	RVS+SHIFT+Q
R	R	SHIFT+R	RVS+R	RVS+SHIFT+R
S	S	SHIFT+S	RVS+S	RVS+SHIFT+S
T	T	SHIFT+T	RVS+T	RVS+SHIFT+T
U	U	SHIFT+U	RVS+U	RVS+SHIFT+U
V	V	SHIFT+V	RVS+V	RVS+SHIFT+V
W	W	SHIFT+W	RVS+W	RVS+SHIFT+W
X	X	SHIFT+X	RVS+X	RVS+SHIFT+X
Y	Y	SHIFT+Y	RVS+Y	RVS+SHIFT+Y
Z	Z	SHIFT+Z	RVS+Z	RVS+SHIFT+Z
[	[	SHIFT++	RVS+[	RVS+SHIFT++
£	£	CBM+-	RVS+£	RVS+CBM+-
]	]	SHIFT+-	RVS+]	RVS+SHIFT+-
↑	↑	CBM+=	RVS+↑	RVS+CBM+=
←	←	CBM+*	RVS+←	RVS+CBM+*
SP	SP	SHIFT+SP	RVS+SP	RVS+SHIFT+SP
!	!	CBM+K	RVS+!	RVS+CBM+K
"	"	CBM+I	RVS+"	RVS+CBM+I
#	#	CBM+T	RVS+#	RVS+CBM+T

TABELLA 6.1				
SIMB.	TASTI DA PREMERE			
	SFONDO	COLORE 1	COLORE 2	COLORE 3
$	$	CBM+@	RVS+$	RVS+CBM+@
%	%	CBM+G	RVS+%	RVS+CBM+G
&	&	CBM++	RVS+&	RVS+CBM++
'	'	CBM+M	RVS+'	RVS+CBM+M
(	(	CBM+£	RVS+(	RVS+CBM+£
)	)	SHIFT+£	RVS+)	RVS+SHIFT+£
*	*	CBM+N	RVS+*	RVS+CBM+N
+	+	CBM+Q	RVS++	RVS+CBM+Q
,	,	CBM+D	RVS+,	RVS+CBM+D
–	–	CBM+Z	RVS+–	RVS+CBM+Z
.	.	CBM+S	RVS+.	RVS+CBM+S
/	/	CBM+P	RVS+/	RVS+CBM+P
0	0	CBM+A	RVS+0	RVS+CBM+A
1	1	CBM+E	RVS+1	RVS+CBM+E
2	2	CBM+R	RVS+2	RVS+CBM+R
3	3	CBM+W	RVS+3	RVS+CBM+W
4	4	CBM+H	RVS+4	RVS+CBM+H
5	5	CBM+J	RVS+5	RVS+CBM+J
6	6	CBM+L	RVS+6	RVS+CBM+L
7	7	CBM+Y	RVS+7	RVS+CBM+Y
8	8	CBM+U	RVS+8	RVS+CBM+U
9	9	CBM+O	RVS+9	RVS+CBM+O
:	:	SHIFT+@	RVS+:	RVS+SHIFT+@
;	;	CBM+F	RVS+;	RVS+CBM+F
<	<	CBM+C	RVS+<	RVS+CBM+C
=	=	CBM+X	RVS+=	RVS+CBM+X
>	>	CBM+V	RVS+>	RVS+CBM+V
?	?	CBM+B	RVS+?	RVS+CBM+B

6.6 CARATTERI MULTICOLORE

Questo paragrafo tratta dell'ultimo modo che hai a disposizione per rappresentare i caratteri con il COMMODORE 16: il modo MULTICOLORE. Come nei modi grafici 3 o 4, nel modo multicolore, ad ogni puntino che forma un carattere corrispondono 2 bit. In questo modo puoi associare ad ogni punto un colore scelto tra 4 possibili (invece che tra 2) ma la matrice di ogni carattere e' 4x8 (anziche' 8x8). Per entrare in modo MULTICOLORE devi porre a 1 il bit 4 del registro 22 del TED (indirizzo 65287 ($FF16)); bisogna cioe' eseguire l'istruzione:

```
POKE 65287, PEEK(65287) OR 16
```

per tornare in modo ad alta risoluzione devi usare l'istruzione:

```
POKE 65287, PEEK(65287) AND 239.
```

Per chiarire le idee vediamo come TED legge il carattere A in modo multicolore sapendo che gli 8 byte che descrivono la A contengono:

```
0 0 0 1 1 0 0 0
0 0 1 1 1 1 0 0
0 1 1 0 0 1 1 0
0 1 1 1 1 1 1 0
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
0 0 0 0 0 0 0 0
```

TED in modo multicolore interpreta ogni byte come una serie di quattro punti, cosi':

```
00 01 10 00   cioe' come:   A B C A
00 11 11 00                 A D D A
01 10 01 10                 B C B C
01 11 11 10                 B D D C
01 10 01 10                 B C B C
01 10 01 10                 B C B C
01 10 01 10                 B C B C
00 00 00 00                 A A A A
```

dove A, B, C e D sono rispettivamente:

- il colore dello schermo.
- il colore 3.
- il colore indicato dal registro 23 (indirizzo 65303 ($FF17)) con la convenzione 16*L+C-1, dove C e' il codice del colore e L la luminosita'.
- il colore proprio del carattere.

I primi 3 colori possono essere scelti tra i 16 del COMMODORE 16, mentre il quarto puo' essere solo uno dei primi 8. Questa piccola limitazione permette, in compenso, di visualizzare contemporaneamente caratteri multicolore e alta risoluzione. TED, infatti, visualizza il carattere come multicolore solo se il byte corrispondente della mappa degli attributi ha il bit 3 a 1 e assume come codice del colore il numero formato dai 3 bit meno significativi. Quando scegli il colore proprio del carattere dovrai sceglierlo tra i primi 8 e aggiungere 8 se lo vuoi multicolore o 0 se lo vuoi normale.
Ecco GRAF9 come esempio di programmazione di un carattere multicolore.

```
0 REM GRAF9
10 PRINTCHR$(147)
20 FORI=0TO7:READA:POKE14336+I,A:NEXT
30 FORI=0TO7:POKE14336+32*8+I,0:NEXT
40 COLOR0,15,0:COLOR4,15,0
50 COLOR1,14,4
60 POKE65303,16*7+1
70 COLOR3,3,3
80 POKE65287,PEEK(65287)OR16
90 POKE65299,(PEEK(65299)AND3)+56
100 POKE65298,PEEK(65298)AND251
110 PRINT"@"
120 GETKEYA$
130 POKE65287,PEEK(65287)AND239
140 POKE65299,(PEEK(65299)AND3)+208
150 POKE65298,PEEK(65298)OR4
1000 DATA165,165,165,165,240,240,240,240
```

COMMENTO A GRAF9

.10: pulisce lo schermo.
.20: pone in memoria i dati relativi al carattere di D/CODE 0.
.30: azzera il carattere di D/CODE 32 (lo spazio).
.40: schermo e bordo blu.
.50: colore del cursore 14 (in modo multicolore vuol dire colore D=6 (verde), carattere da visualizzare in modo multicolore).

.60: colore C=2 (bianco).
.70: colore B=3 (rosso).
.80: entra in modo multicolore.
.90/100: descrizione dei caratteri in 14336 ($3800).
.110: scrive il carattere di D/CODE 0.
.120: attende la pressione di un tasto.
.130: torna al modo alta risoluzione.
.140/150: caratteri in ROM.
.1000: dati relativi al carattere.

Il carattere che abbiamo programmato e' questo:

165 = 10100101 = CCBB
165 = 10100101 = CCBB
165 = 10100101 = CCBB
165 = 10100101 = CCBB
240 = 11110000 = DDAA
240 = 11110000 = DDAA
240 = 11110000 = DDAA
240 = 11110000 = DDAA

In questo modo il carattere programmato e' composto da 4 quadratini di colore diverso. Questo esempio mostra come TED possa porre 4 colori diversi nella posizione di un solo carattere, ma il risultato non e' certamente dei piu' pittoreschi. Prova quindi il programma GRAF10 il cui scopo e' quello di disegnare dei coloratissimi omini spaziali.

```
0 REM GRAF10
10 PRINTCHR$(147)
20 FORI=0TO47:READA:POKE14336+I,A:NEXT
30 FORI=0TO7:POKE14336+32*8+I,0:NEXT
40 COLOR0,1:COLOR4,1
50 COLOR1,16,6
60 POKE65303,16*4+6
70 COLOR3,3,3
80 POKE65287,PEEK(65287)OR16
90 POKE65299,(PEEK(65299)AND3)+56
100 POKE65298,PEEK(65298)AND251
110 CHAR,0,10," @A @A @A @A @A @A @A"
115 PRINT" @A @A @A @A @A @A"
120 PRINT" BC BC BC BC BC BC BC";
125 PRINT" BC BC BC BC BC BC"
130 PRINT" DE DE DE DE DE DE DE";
135 PRINT" DE DE DE DE DE DE"
140 GETKEYA$
150 POKE65287,PEEK(65287)AND239
160 POKE65299,(PEEK(65299)AND3)+208
170 POKE65298,PEEK(65298)OR4
1000 DATA192,192,49,53,21,81,65,85
```

```
1010 DATA3,3,76,92,84,69,65,85
1020 DATA84,85,22,5,3,5,21,85
1030 DATA21,85,148,80,192,80,84,85
1040 DATA85,85,21,5,1,34,42,10
1050 DATA85,85,84,80,64,136,168,160
```

COMMENTO A GRAF10

.10: pulisce lo schermo.
.20: programma i caratteri che formano l'omino.
.30: programma lo spazio.
.40: schermo e bordo neri.
.50: pone a 16 il colore dei caratteri (il colore D); saranno quindi caratteri multicolore con colore 8, cioe' giallo.
.60: pone a 7 (blu) il colore C.
.70: pone a 3 (rosso) il colore B.
.80: entra in modo multicolore.
.90/100: descrizione dei caratteri in 14336 ($3800).
.110/135: scrive partendo dalla colonna 0, riga 10, i caratteri che compongono gli omini.
.140: attende la pressione di un tasto.
.150: torna al modo normale.
.160/170: riporta la descrizione dei caratteri in ROM.
.1000/1050: dati relativi agli omini.

6.7 ANNULLAMENTO DELLO SCHERMO

E' possibile annullare tutto cio' che compare sullo schermo ponendo a 0 il bit 4 del registro 6 del TED. Tale registro risponde all'indirizzo 65286 ($FF06); l'istruzione da impartire e' quindi:

POKE 65286, PEEK(65286) AND 239

eseguendola lo schermo diventa completamente vuoto e dello stesso colore del bordo, come quando il calcolatore accede a nastro. Per riportare la situazione alla normalita' devi compiere l'operazione inversa, cioe' eseguire:

POKE 65286, PEEK(65286) OR 16.

Puoi sfruttare questa opzione di TED per compiere del-

le operazioni sullo schermo mentre questo non si puo' vedere, ma il vero significato di questo bit e' un altro. Come sai TED chiede del tempo alla CPU per poter leggere dalla memoria i dati che gli servono per sapere cosa visualizzare. Quando lo schermo viene annullato, TED non ha piu' bisogno di quei dati e quindi "lascia in pace" la CPU che puo' svolgere le sue funzioni con una velocita' notevolmente piu' alta. Prova ora a far girare il programma GRAF11:

```
0 REM GRAF11
10 POKE65286,PEEK(65286)AND239
20 TI$="000000"
30 FORI=1TO10000:NEXT
40 A=INT(TI/6)/10
50 POKE65286,PEEK(65286)OR16
60 TI$="000000"
70 FORI=1TO10000:NEXT
80 B=INT(TI/6)/10
90 PRINT"CON SCHERMO ANNULLATO :"A"SECONDI"
100 PRINT"CON SCHERMO NON ANNULLATO :"B"SECONDI"
```

Come vedi il tempo impiegato a schermo annullato e' sensibilmente minore di quello impiegato nello stato normale. Ma il vantaggio piu' importante non e' nella velocita' che si guadagna, ma nella precisione con cui la CPU puo' calcolare i ritardi. Con adatte routine in linguaggio macchina, infatti, si possono calcolare ritardi con una precisione di meno di 1 milionesimo di secondo a patto di mascherare gli interrupt e di annullare lo schermo. Ritardi cosi' precisi servono in alcune operazioni di I/O come quelle del nastro.

6.8 SCORRIMENTO FINE (SMOOTH SCROLLING) E REGISTRO DI LINEA

TED permette di far scorrere tutto il contenuto dello schermo di un solo punto (un ottavo di carattere), sia in direzione orizzontale che in direzione verticale.
I registri con cui si controllano queste operazioni sono rispettivamente i registri 7 e 6 che rispondono agli indirizzi 65287 ($FF07) e 65286 ($FF06). I bit 2-0 di questi registri indicano quale delle 8 possibili posizioni (da 0 a 7) devono assumere i caratteri

sul video, mentre il bit 3 seleziona, se posto a 0, rispettivamente il modo a 38 colonne (1 per 40 colonne) e a 24 righe (1 per 25 righe).
Nota che lo schermo rimane sempre di 25 righe di 40 colonne, ma ponendo a 0 questi bit vengono visualizzate solamente 24 righe e 38 colonne: questo permette di aggiungere, nella parte nascosta dello schermo, i nuovi dati e farli entrare lentamente nella parte visibile realizzando lo scorrimento fine. Per ottenere uno scorrimento fine, quindi, devi:

. diminuire la parte visibile dello schermo nella direzione in cui lo vuoi far scorrere (ad esempio, se vuoi far scorrere lo schermo in direzione verticale, lo dovrai portare a 24 righe).
. porre i 3 bit meno significativi del registro interessato a 7 o a 0 (se lo scorrimento viene fatto dall'alto verso il basso a 0, viceversa a 7; da sinistra a destra a 0, viceversa a 7).
. scrivere i nuovi dati da visualizzare nella parte nascosta dello schermo.
. incrementare (o decrementare) il contenuto dei bit 2-1 del registro interessato fino al valore massimo (o minimo).
. riportare il contenuto dei bit 2-0 al valore del secondo passo e far scorrere tutta la scritta di un carattere intero nella direzione dello scorrimento. Questa operazione deve essere fatta con una routine in linguaggio macchina poiche' in BASIC si vede nettamente che questa operazione equivale a fare 7 passi indietro piu' 8 in avanti, mentre deve sembrare che sia un solo passo in avanti.
. tornare al punto 3.

Vediamo un paio di esempi che chiariscono meglio questi concetti:

```
0 REM GRAF12
10 COLOR0,15,1:COLOR1,14,6:COLOR4,15,1
20 PRINTCHR$(147)
30 PRINT"CHE SCRITTA VUOI FAR SCORRERE ":PRINT
40 GETKEYA$:IFA$=CHR$(13)ORLEN(B$)=253THEN60
50 B$=B$+A$:PRINTA$;:GOTO40
60 B$=B$+"   "
70 L=LEN(B$):PRINTCHR$(147)
80 POKE65287,PEEK(65287)AND247
90 FORI=1TOLEN(B$)
100 PRINTCHR$(19)" ";
110 PRINTCHR$(20);:POKE65287,(PEEK(65287)AND248)+7
```

```
120 PRINTCHR$(17)CHR$(157)MID$(B$,I,1)
130 FORJ=6TO0STEP-1
140 POKE65287,(PEEK(65287)AND248)+J
150 NEXT J
160 NEXT I
170 GOTO90
```

COMMENTO A GRAF12

.10/30: inizializza il video.
.40/50: riceve una stringa che puo' essere lunga fino a 253 caratteri.
.60: aggiunge 2 spazi alla stringa.
.70/80: calcola la lunghezza della stringa, pulisce lo schermo e seleziona il modo a 38 colonne.
.90: inizializza un ciclo di tanti passi quanti sono i caratteri che formano la stringa.
.100: posiziona il cursore sulla seconda colonna della prima riga.
.110: cancella un carattere in modo da far scorrere tutta la prima riga a sinistra e riportare il cursore nell'angolo in alto a sinistra e pone 7 nei bit 2¬0 del registro dello scorrimento orizzontale (7 = massimo a destra).
.120: sposta il cursore in basso e quindi a sinistra per posiazionarlo sull'ultima colonna della prima riga dove stampa l'I¬esimo carattere della stringa.
.130/150: ciclo che decrementa il contenuto dei bit 2¬0 del registro dello scorrimento orizzontale fino a 0, spostando le scritte gradualmente a sinistra.
.160: prossimo carattere della stringa.
.170: torna alla linea 90 per cominciare nuovamente dall'inizio della stringa.

Facendo girare GRAF12 ti accoregerai che un programma completamente BASIC non da' dei buoni risultati. La linea 110 andra' quindi sostituita con un'adeguata routine in linguaggio macchina come nel programma GRAF13 in cui la linea 110 e' stata sostituita dalla routine GRAF14. GRAF14 usa un registro di TED che non abbiamo ancora illustrato: il REGISTRO DI LINEA. Questo registro indica in ogni istante quale delle 313 linee che formano lo schermo viene disegnata da TED; risponde all'indirizzo 65309 ($FF1D) per gli 8 bit bassi e da' il bit 8 (servono infatti 9 bit, da 0 a 8, per esprimere un numero tra 0 e 312) nel bit meno significativo se l'indirizzo e' 65308 ($FF1C). Le linee che formano la parte in cui TED puo' visua-

lizzare i caratteri vanno dalla 4 alla 203. GRAF14, prima di eseguire cio' che veniva svolto dalla linea 110 in GRAF12, controlla che il registro di linea contenga $CC (204) cioe' si assicura di svolgere le sue funzioni mentre TED sta disegnando la parte inferiore del bordo. Questo accorgimento e' necessario perche', anche se la routine in linguaggio macchina e' velocissima, provocherebbe gli stessi effetti del BASIC se svolgesse le sue funzioni mentre TED sta disegnando le linee 4-11, su cui essa agisce.
Vediamo ora i listati di GRAF13 e GRAF14.

```
0 REM GRAF13
5 POKE56,63:POKE55,233:CLR
6 FORI=0TO22:READA:POKE16361+I,A:NEXT
10 COLOR0,15,1:COLOR1,14,6:COLOR4,15,1
20 PRINTCHR$(147)
30 PRINT"CHE SCRITTA VUOI FAR SCORRERE ":PRINT
40 GETKEYA$:IFA$=CHR$(13)ORLEN(B$)=253THEN60
50 B$=B$+A$:PRINTA$;:GOTO40
60 B$=B$+"  "
70 L=LEN(B$):PRINTCHR$(147)
80 POKE65287,PEEK(65287)AND247
90 FORI=1TOLEN(B$)
100 PRINTCHR$(19)" ";
110 SYS16361
120 PRINTCHR$(17)CHR$(157)MID$(B$,I,1)
130 FORJ=6TO0STEP-1:FORK=1TO20:NEXT
140 POKE65287,(PEEK(65287)AND248)+J
150 NEXT J
160 NEXT I
170 GOTO90
180 DATA120,173,29,255,201,204,208,249
190 DATA169,20,32,210,255,173,7,255
200 DATA9,7,141,7,255,88,96
```

```
MONITOR

   PC  SR AC XR YR SP
; 0000 00 00 00 00 F8

. 3FE9  78          SEI
. 3FEA  AD 1D FF    LDA $FF1D
. 3FED  C9 CC       CMP #$CC
. 3FEF  D0 F9       BNE $3FEA
. 3FF1  A9 14       LDA #$14
. 3FF3  20 D2 FF    JSR $FFD2
. 3FF6  AD 07 FF    LDA $FF07
. 3FF9  09 07       ORA #$07
. 3FFB  8D 07 FF    STA $FF07
. 3FFE  58          CLI
. 3FFF  60          RTS
```

Il programma GRAF13 e' identico a GRAF12 tranne nelle linee:
.5: che abbassa i puntatori di memoria e pone la routine GRAF14 da 16361 ($3FE9) a 16383 ($3FFF).
.110: che salta alla routine in linguaggio macchina.

Vediamo ora come funziona la routine GRAF14, con riferimento agli indirizzi esadecimali.

.3FE9: maschera gli interrupt per poter leggere continuamente il registro di linea.
.3FEA: carica nell'accumulatore i bit 7-0 del registro di linea.
.3FED: controlla se e' arrivato a $CC, non controlla il bit 8 perche' questo non puo' che essere 0 quando 7-0 contengono $CC: infatti il registro di linea puo' contenere da $00 a $138.
.3FEF: se il registro di linea non contiene ancora $CC torna a leggere.
.3FF1: carica nell'accumulatore $14 (20, che e' il codice ASCII di DELETE).
.3FF3: lo stampa.
.3FF6/3FFB: pone a 1 i bit 0-2 del registro di scorrimento orizzontale.
.3FFE: sente nuovamente gli interrupt.
.3FFF: torna al BASIC.

Per uno scorrimento fine verticale le cose si complicano ulteriormente: infatti se chiediamo uno scorrimento fine mentre il registro di linea e' in una posizione compresa tra la linea 4 e la 203, otteniamo uno spiacevole sfarfallio dello schermo e lo stesso vale, a maggior ragione, quando chiediamo un'operazione del tipo 7 passi indietro piu' 8 avanti. Per risolvere questi problemi bisogna quindi sincronizzare gli scorrimenti fini col registro di linea e compiere lo scorrimento degli 8 passi in maniera molto rapida. Purtroppo neanche una routine in linguaggio macchina riesce ad essere cosi' veloce da far scorrere lo schermo mentre il registro di linea e' tra 204 e 3. Abbiamo quindi fatto una routine in linguaggio macchina, la GRAF16, che usa 3 mappe video e degli attributi. Nella solita coppia, da $0800 a $0BE8 e da $0C00 a $0FE8, poniamo, con l'istruzione BASIC CHAR, le nuove stringhe; le altre due coppie, che sono visualizzate alternativamente da TED, occupano le locazioni da $3000 a $33E8 e da $3400 a $37E8 la prima

coppia, da $3800 a $3BE8 e da $3C00 a $3FE8 la seconda. In questo modo, quando TED mostra una coppia, vengono trasferiti i dati della mappa video e attributi normali nell'altra. Al fatidico momento degli 8 passi avanti e 7 indietro, non facciamo altro che cambiare mappa video e attributi e compiere 7 passi indietro (operazione sufficentemente veloce): potremo quindi, con calma, aggiungere i nuovi dati nelle mappe normali e trasferire il tutto nelle mappe non visualizzate. Vediamo, nel dettaglio, come funziona GRAF16:

```
MONITOR

   PC  SR AC XR YR SP
; 0000 00 00 00 00 F8

. 2FB0  78        SEI
. 2FB1  AD 1D FF  LDA $FF1D
. 2FB4  C9 CC     CMP #$CC
. 2FB6  D0 F9     BNE $2FB1
. 2FB8  AD FF 2F  LDA $2FFF
. 2FBB  D0 0D     BNE $2FCA
. 2FBD  AD 14 FF  LDA $FF14
. 2FC0  49 08     EOR #$08
. 2FC2  8D 14 FF  STA $FF14
. 2FC5  A9 08     LDA #$08
. 2FC7  8D FF 2F  STA $2FFF
. 2FCA  CE FF 2F  DEC $2FFF
. 2FCD  AD 06 FF  LDA $FF06
. 2FD0  29 F0     AND #$F0
. 2FD2  18        CLC
. 2FD3  6D FF 2F  ADC $2FFF
. 2FD6  8D 06 FF  STA $FF06
. 2FD9  60        RTS
. 2FDA  AD 14 FF  LDA $FF14
. 2FDD  29 F8     AND #$F8
. 2FDF  49 08     EOR #$08
. 2FE1  85 04     STA $04
. 2FE3  A9 08     LDA #$08
. 2FE5  85 06     STA $06
. 2FE7  A0 00     LDY #$00
. 2FE9  84 03     STY $03
. 2FEB  84 05     STY $05
. 2FED  B1 05     LDA ($05),Y
. 2FEF  91 03     STA ($03),Y
. 2FF1  C8        INY
. 2FF2  D0 F9     BNE $2FED
. 2FF4  E6 04     INC $04
. 2FF6  E6 06     INC $06
. 2FF8  A5 06     LDA $06
. 2FFA  C9 10     CMP #$10
. 2FFC  D0 EF     BNE $2FED
. 2FFE  60        RTS
```

I byte usati da questo programma sono:

.$2FFF: POSIZIONE in cui TED visualizza lo schermo

(da 0 a 7).
.$03,$04: PUNTATORE 1 che indica l'indirizzo di partenza delle mappe in cui vanno trasferiti i dati.
.$05,$06: PUNTATORE 2 che indica l'indirizzo di partenza delle mappe normali.

GRAF16 e' composto da due routine: la prima fa scorrere lo schermo di una linea ogni volta che viene chiamata e, se necessario, cambia la pagina visualizzata; la seconda (che verra' chiamata dal BASIC dopo aver inserito i nuovi dati nelle mappe normali), trasferisce i dati dalle mappe normali a quelle non visualizzate.
Ecco lo schema a blocchi di GRAF16:

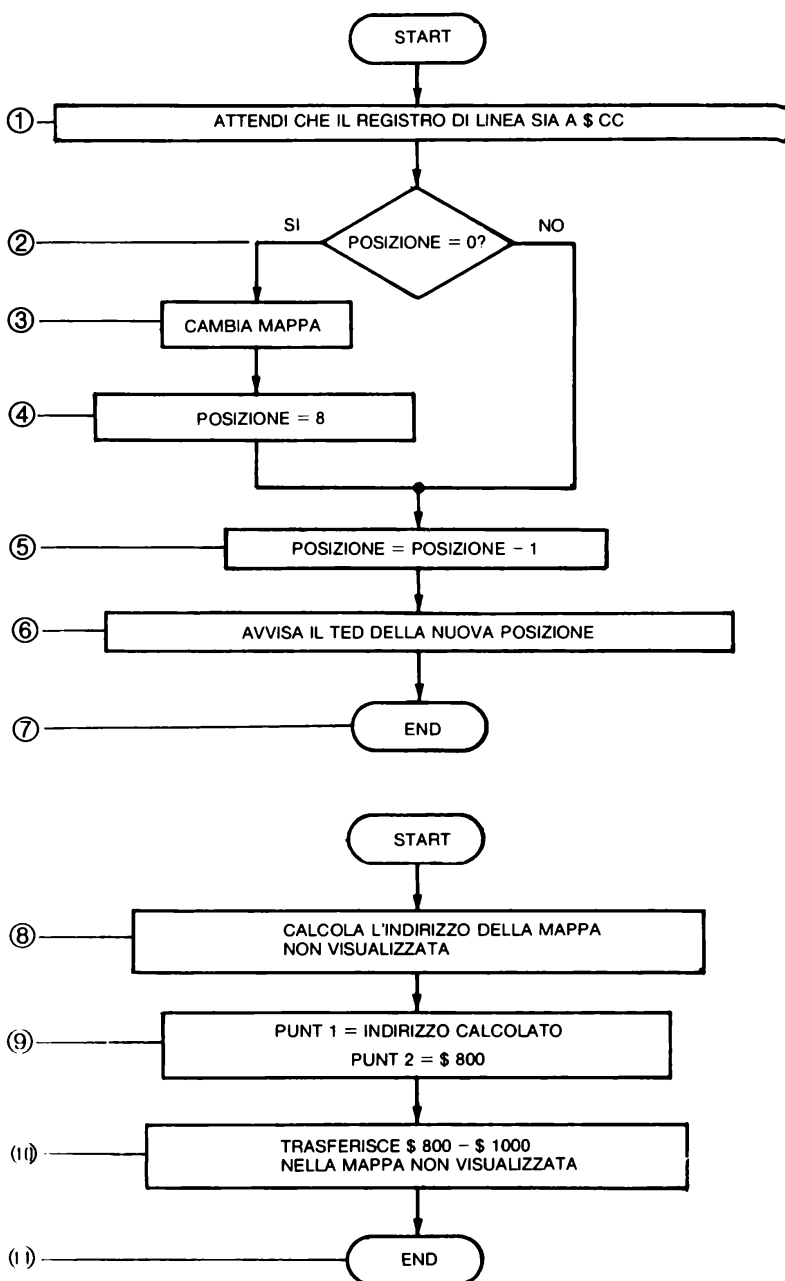



Figura 6.1 Diagramma a blocchi sottoprogramma GRAF16

Ecco infine la corrispondenza tra i blocchi dello schema e le linee del programma, riferendoci agli indirizzi esadecimali:

BLOCCO 1 : linee 2FB0-2FB6.
BLOCCO 2 : linee 2FB8-2FBB.
BLOCCO 3 : linee 2FBD-2FC2.
BLOCCO 4 : linee 2FC5-2FC7.
BLOCCO 5 : linee 2FCA-2FCD.
BLOCCO 6 : linee 2FD0-2FD6.
BLOCCO 7 : linee 2FD9.
BLOCCO 8 : linee 2FDA-2FDF.
BLOCCO 9 : linee 2FE1-2FEB.
BLOCCO 10 : linee 2FED-2FFC.
BLOCCO 11 : linee 2FFE.

Vediamo ora come il BASIC gestisce queste due routine:

```
0 REM GRAF15
10 POKE56,47:POKE55,176:CLR:TRAP500
30 FORI=0TO78:READA:POKE12208+I,A:NEXT
50 READN
60 DIMCL(N-1),LU(N-1),LN$(N-1)
70 FORI=0TON-1
80 READCL(I),LU(I),LN$(I)
110 NEXT
120 COLOR0,1:COLOR4,1:SCNCLR
130 POKE65300,48:SYS12250
140 POKE65300,56:SYS12250
170 POKE12287,7
175 DO
180 FORI=0TON-1
190 COLOR1,CL(I),LU(I):CHAR,0,24,CHR$(17)+LN$(I)
210 SYS12250
220 SYS12208
230 FORJ=1TO7
240 FORK=1TO60:NEXT
250 SYS12208
260 NEXTJ
270 NEXTI
280 LOOP
500 COLOR1,2,7:POKE65286,27:POKE65300,15
510 PRINT:PRINTERR$(ER):PRINT"IN"EL:END
1000 DATA!20,173,29,255,201,204,208,249
1010 DATA173,255,47,208,13,173,20,255
1020 DATA73,8,141,20,255,169,8,141
1030 DATA255,47,206,255,47,173,6,255
1040 DATA41,240,24,109,255,47,141,6
1050 DATA255,96,173,20,255,41,248,73
1060 DATA8,133,4,169,8,133,6,160
1070 DATA0,132,3,132,5,177,5,145
1080 DATA3,200,208,249,230,4,230,6
1090 DATA165,6,201,16,208,239,96
```

```
10000 DATA10
10010 DATA3,3,"
10020 DATA5,4,"
10030 DATA4,4,"
10040 DATA6,4,"
10050 DATA9,5,"
10060 DATA13,4,"
10070 DATA8,5,"
10080 DATA1,1,""
10090 DATA1,1,""
10100 DATA1,1,""
```

COMMENTO A GRAF15

.10: pone la fine della memoria in 12208 ($2FB0) dove inizia il programma in linguaggio macchina, in $3000 inizia la prima mappa attributi, in $3400 la prima mappa video, in $3800 inizia la seconda mappa attributi, in $3C00 la seconda mappa video. Indica che la routine di errore parte dalla linea 500.
.30: pone in memoria GRAF16.
.50: legge il numero di stringhe che dovra' visualizzare.
.60: dimensiona 3 vettori di tanti elementi quante sono le stringhe: per ogni stringa memorizza il colore, la luminosita' e la stringa stessa.
.70/110: riempe i vettori.
.120: schermo e sfondo neri, pulisce lo schermo.
.130: mappa attributi in $3000 (e quindi mappa video in $3400); salta alla routine che trasferisce nell'altra coppia il contenuto delle mappe normali (che al momento sono vuote).
.140: mappa attributi in $3800 (e quindi mappa video in $3C00); salta alla stessa routine pulendo anche l'altra coppia di mappe.
.170: pone 7 nel byte che indica la posizione del video alla routine in linguaggio macchina ($2FFF).
.175: inizializza un ciclo senza fine che si chiude in 280: per uscire dal programma basta premere RUN/STOP.
.180: per ogni stinga da visulizzare esegue da qui a 260.
.190: seleziona colore e luminosita' richiesti; stampa la stringa sulla mappa normale provocando uno scorrimento (con CHR$(17)).
.210: salta alla routine che cambia mappe e trasferisce le mappe normali nelle mappe non visualizzate.
.220: salta alla routine che provoca lo scorrimento fine.

.230/260: provoca i 7 rimanenti scorrimenti fini.
.270: prossima stringa.
.280: chiude il ciclo aperto in 175.
.500/510: abbiamo programmato una routine di errore perche' il SISTEMA OPERATIVO segnala l'errore sulla mappa normale; la routine di errore, invece, lo segnala dopo aver riportato le mappe in posizione normale e aver riportato lo schermo a 25 colonne, e arresta il programma.
.1000/1090: dati relativi alla routine GRAF16.
.10000: le 10 stringhe da visualizzare.
.10010/10100: colori, luminosita' e caratteri di ogni stringa.

Puoi ovviamente cambiare le ultime 11 linee del programma e far scorrere il messaggio che preferisci.

6.9 RIEPILOGO

In questo paragrafo abbiamo riassunto le istruzioni necessarie per sfruttare le capacita' di TED che hai visto in questo capitolo: quando ti sarai impadronito delle tecniche che ti permettono di sfruttare in pieno TED, ti bastera' consultare queste pagine per ricordarti quali sono i registri che ti interessano.

- MAPPA VIDEO E ATTRIBUTI: per cambiare l'indirizzo di partenza della mappa video e attributi devi dare le istruzioni:

  ```
  POKE 65300,N
  ```

  oppure

  ```
  LDA #$N
  STA $FF14
  ```

  dove N*256 e' l'indirizzo del primo byte della nuova mappa attributi, la nuova mappa video sara' automaticamente posta 1024 byte oltre. N deve essere un numero divisibile per 8: se non lo e' TED considera (N/8)*8 (cioe' non considera i bit 2-0). Normalmente la mappa attributi e' posta in $0800 e quindi il numero N necessario a riportare il calcolatore nello stato normale e' 8.

DESCRIZIONE DEI CARATTERI: per cambiare l'indirizzo del primo byte di descrizione dei caratteri devi dare le istruzioni:

```
POKE 65298,PEEK(65298) AND 251

POKE 65299,(PEEK(65299) AND 3) + N
```

oppure

```
LDA $FF12
AND #$FB
STA $FF12
LDA $FF13
AND #$03
ORA #$N
STA $FF13
```

Dove N*256 e' l'indirizzo desiderato e N e' divisibile per 4 (pena il malfunzionamento del sistema). Per riportare la mappa dei caratteri in ROM devi sostituire l'operazione AND 251 (AND #$FB) con OR 4 (ORA #$04) e N vale 208 (#$D0)

CARATTERI A SFONDO PROGRAMMABILE: per entrare in modo sfondo programmabile devi dare le istruzioni:

```
POKE 65286,PEEK(65286) OR 64
POKE 65302,A
POKE 65303,B
POKE 65304,C
```

oppure

```
LDA $FF06
ORA #$40
STA $FF06
LDA #$A
STA $FF16
LDA #$B
STA $FF17
LDA #$C
STA $FF18
```

dove A, B e C indicano i colori e le luminosita' dello sfondo 1, 2 e 3 rispettivamente e sono codificati come L*16+K-1, con L luminosita' e K colore.
Per uscire dal modo sfondo programmabile devi dare le istruzioni:

```
POKE 65286,PEEK(65286) AND 191
```

oppure

```
LDA $FF06
AND #$BF
STA $FF06
```

MODO MULTICOLORE: per entrare in modo multicolore devi dare le istruzioni:

```
POKE 65287,PEEK(65287) OR 16
COLOR1,A1,A2
COLOR3,B1,B2

POKE 65303,C1*16+C2-1
```

oppure

```
LDA $FF07
ORA #$10
STA $FF07
LDA #$B1*16+B2-1
STA $FF16
LDA #$C1*16+C2-1
STA $FF17
```

dove A1, B1 e C1 sono i codici delle luminosita' desiderate e A2, B2 e C2 i codici dei colori desiderati. Nelle routine in linguaggio macchina, devi porre il numero A1*16+A2-1 nel byte della mappa degli attributi corrispondente al carattere che vuoi colorare cosi'. Ricorda che A2 deve essere maggiore di 7 altrimenti il carattere verra' visualizzato in modo alta risoluzione.

ANNULAMENTO DELLO SCHERMO: per ottenere l'annullamento dello schermo ti basta dare le istruzioni:

```
POKE 65286,PEEK(65286) AND 239
```

oppure

```
LDA $FF06
AND #$EF
STA $FF06
```

per tornare allo schermo normale basta cambiare l'operazione AND 239 (AND #$EF) con OR 64 (ORA #$40).

CAPITOLO 7

TECNICHE DI PROGRAMMAZIONE E ESEMPI

7.1 INTRODUZIONE

In questo capitolo riportiamo alcuni esempi di programmi, discutendone l'impostazione e commentandoli.

7.2 DIVISIONE PAROLE IN SILLABE

Alcuni programmi di ELABORAZIONE TESTI (Word Processing) producono dei file di testo di tipo sequenziale ASCII, che contengono il testo, ma non provvedono automaticamente alla separazione delle parole in sillabe con il trattino fantasma, da utilizzare eventualmente in fase di stampa per andare a capo.
Abbiamo preparato il programma HYPHEN (trattino in inglese) per ottenere di leggere un file di testo, prodotto dal programma EASY SCRIPT del COMMODORE 64, e produrre un file di testo modificato, con altro nome e con le parole divise in due parti da un trattino fantasma (SHIFT-@), riconosciuto come tale nella fase di stampa da EASY SCRIPT.

Per separare in due parti le parole abbiamo usato un algoritmo empirico. Abbiamo osservato che e' possibile dividere le lettere che compongono le parole in tre categorie:

- Vocali o mute-liquide-nasali (a,e,i,o,u,h,l,m,n,r)
- Consonanti mute-liquide-nasali (h,l,m,n,r)
- Consonanti dure (b,c,d,f,g,p,q,s,t,v,z)

Abbiamo creato tre vettori di variabili numeriche che consideriamo booleane: LL, VV, CC. Ognuno di questi 3 vettori ha 255 elementi, uno per ogni codice ASCII. Il

vettore VV indica quali lettere sono VOCALI o LIQUIDE: VV(65), ad esempio, viene posto a vero (-1), perche' la lettera A, che ha codice ASCII 65, e' vocale. Ugualmente il vettore LL indica quali lettere sono MUTE-LIQUIDE-NASALI, e il vettore CC indica le lettere che sono consonanti dure. In conseguenza gli elementi dei 3 vettori contengono o 0 o -1.
Per effettuare la separazione in sillabe osserviamo un gruppo di tre lettere e poniamo la divisione tra il primo e il secondo dei tre caratteri che analizziamo, a patto che si verifichi una delle seguenti condizioni:

1- Le prime due lettere sono uguali e sono consonanti (liquide-nasali o dure): e' il caso della doppia.
2- Non separiamo se la prima lettera e' S: dopo la S non si separa, se non per la doppia, gia' considerata.
3- Vocale o liquida - S - qualunque lettera esclusa S
4- Vocale o liquida - consonante dura - vocale o liquida

5- Vocale o liquida - liquida - vocale

```
1 REM HYPHEN
3 SH$="-":REM TRATTINO FANTASMA
5 REM VARIABILI DI LAVORO
7 DIMLL(255),VV(255),CC(255)
9 DIMA$(3),L(3),V(3),C(3)
11 FORI=1TO5:READA$:VV(ASC(A$))=-1:NEXTI
13 FORI=6TO10:READA$:VV(ASC(A$))=-1
15 LL(ASC(A$))=-1:NEXTI
17 FORI=11TO21:READA$:CC(ASC(A$))=-1:NEXTI
19 REM RICHIESTA NOMI FILE INPUT E OUTPUT
21 INPUT"FILE INPUT";FI$
23 INPUT"FILE OUTPUT";FO$
25 REM APERTURA FILE
27 OPEN2,8,2,FI$:OPEN3,8,3,FO$+",S,W"
29 REM CICLO DI LETTURA DI UNA PAROLA
31 A$=""
33 GET#2,B$:RS=ST
35 IFB$<>" "ANDB$<>CHR$(13)THENA$=A$+B$:GOTO47
37 GOSUB53:REM VA A METTERE TRATTINO
39 IFRS=0THENPRINT#3,B$;:PRINTB$;:GOTO31
41 REM CHIUDE SE FINITO FILE DI INPUT
43 CLOSE3:CLOSE2
45 END
47 IFRS=0THEN33
49 GOTO37
51 REM ROUTINE CHE DIVIDE CON TRATTINO
53 IFLEN(A$)<4THENPRINTA$;:PRINT#3,A$;:RETURN
55 FORI=1TOLEN(A$)-2
```

```
57 A$(1)=MID$(A$,I,1)
59 A$(2)=MID$(A$,I+1,1)
61 A$(3)=MID$(A$,I+2,1)
63 PRINTA$(1);:PRINT#3,A$(1);:GOSUB91
65 IFA$(1)=A$(2)ANDNOT(V(1)ANDNOTL(1))THEN77
67 IFA$(1)="S"THEN79
69 IFV(1)ANDA$(2)="S"ANDA$(3)<>"S"THEN77
71 IFV(1)ANDC(2)ANDV(3)THEN77
73 IFV(1)ANDL(2)AND(V(3)ANDNOTL(3))THEN77
75 GOTO79
77 IFI>LEN(A$)/2.5THEN83
79 NEXTI:PRINTA$(2);A$(3);
81 PRINT#3,A$(2)A$(3);:RETURN
83 PRINT"-";:PRINT#3,SH$;
85 PRINTMID$(A$,I+1,254);
87 PRINT#3,MID$(A$,I+1,254);:RETURN
89 REM ROUTINE CHE PRELEVA 3 CARATTERI
91 FORJ=1TO3
93 C(J)=CC(ASC(A$(J)))
95 V(J)=VV(ASC(A$(J)))
97 L(J)=LL(ASC(A$(J)))
99 NEXT:RETURN
101 REM VOCALI E CONSONANTI
103 REM VOCALI
105 DATA A,E,I,O,U
107 REM MUTE, LIQUIDE E NASALI (LIQUIDE)
109 DATA H,L,M,N,R
111 REM LABIALI, DENTALI, GUTTURALI (CONSONANTI)
113 DATA B,C,D,F,G,P,Q,S,T,V,Z
```

COMMENTO A HYPHEN

.3: La variabile SH$ contiene il carattere da porre nel file di testo nei punti dove si puo' dividere una parola. Il carattere tra virgolette e' SHIFT-@ per il programma EASY SCRIPT del COMMODORE 64.

.5/17: Dimensiona e riempie i tre vettori booleani.

.19/27: Chiede i nomi dei files e li apre, uno in lettura, l'altro in scrittura.

.29/39 e 47/49: Riceve in A$ una parola dal file in ingresso e la passa al sottoprogramma in 53. Queste linee vengono ripetute fino alla fine del file di ingresso.

.41/45: Chiude i file e termina

.53: Se A$ ha meno di 4 caratteri non viene separata.

.55/99: Ciclo che viene eseguito una volta per ogni carattere di A$, meno che per gli ultimi due caratteri.

.65: Verifica condizione 1.

.67: Verifica condizione 2.

.69: Verifica condizione 3.

.71: Verifica condizione 4.

.73: Verifica condizione 5.
.75: Se non si e' verificata nessuna delle condizioni, la parola non viene separata con il trattino fantasma.
.77: Se e' stata superata la meta' della parola, viene aggiunto nel file SH$ (il trattino fantasma) e stampato sul video un trattino. Una parola puo' contenere un solo trattino fantasma. Puoi cambiare questa linea e porre tutti i trattini fantasma possibili nella parola, se il tuo programma di stampa non si ferma al primo, come EASY SCRIPT del COMMODORE 64.
.87: Chiude il ciclo iniziato a 55 e ritorna dal sottoprogramma alla fine del ciclo.
.91/99: Aggiorna i valori di tre vettori booleani temporanei in funzione dei tre caratteri da analizzare, ponendo, ad esempio, a C(2)=-1 se il secondo dei tre caratteri in questione e' consonante dura.
.101/113: Linee DATA che contengono le lettere. Non scambiare ne' l'ordine delle linee ne' dei caratteri.

7.3 DISEGNO DEI CARATTERI

Alle pagine 240 e 241, Figura B.4, del primo volume sono riportati i caratteri ingranditi, facendo corrispondere un asterisco ad ogni puntino del carattere, cioe' ad ogni bit 1 della sua descrizione.
La figura e' stampata dal programma DISECAR, che riportiamo in questo paragrafo.
Il problema consiste nell'andare a prelevare dalla ROM la descrizione di un carattere che ha un determinato codice, espresso in D/CODE. Il COMMODORE 16 ha memorizzate in ROM le descrizioni dei 128 caratteri del set maiuscolo/grafico e dei 128 caratteri del set minuscolo/maiuscolo. Queste descrizioni occupano in tutto 2K byte (2048 byte). I caratteri in campo inverso si ottengono scambiando i bit 0 con bit 1 e viceversa. La difficolta' consiste nel fatto che non si puo' accedere dal BASIC alla ROM che contiene le descrizioni, che si trovano dal byte 53248 (D000H) al byte 55295 (D7FFH). Per questa ragione, prima di caricare in memoria il programma DISECAR, si devono trasferire dalla ROM in RAM le descrizioni dei caratteri, con il comando MONITOR e la funzione T di trasferimento. Per lavorare si deve procedere cosi':

.1) eseguire in modo immediato:
POKE 55,0:POKE 56,55
per abbassare il TOP della memoria dedicata al BASIC a 14079 (36FFH).

.2) eseguire in modo immediato:
MONITOR
T D000 D7FF 3700
che trasferisce i 2048 byte che stanno da D000 a D7FF in quelli che iniziano in 3700 (che corrisponde all'indirizzo decimale 14080). L'ultimo byte occupato risulta quello di indirizzo 16127 (3EFFH).

.3) caricare ed eseguire il programma DISECAR.

Il programma chiede se vuoi ottenere il risultato su video o stampante, poi chiede il set di caratteri che desideri e il codice del carattere in D/CODE, tra 0 e 255. Se il codice supera 127, esso viene diminuito di 128 e viene posto a 1 lo switch SW, per ricordare che il carattere deve essere in campo inverso.
Viene poi calcolato il valore del puntatore per prelevare gli 8 byte della descrizione del carattere. Viene stampata un'intestazione e poi il carattere ingrandito usando degli asterischi, il valore binario e decimale di ogni byte della descrizione. Se SW=1 vengono invertiti i bit della descrizione per ottenere il campo inverso.
Il programma termina dopo aver stampato un risultato, per proseguire devi scrivere ancora RUN.
Ricorda dopo aver eseguito il programma di rimettere al valore usuale i byte 55 e 56, oppure di eseguire un RESET.

```
0 REM DISECAR
1 INPUT"■■RISULTATI SU STAMPANTE? (S/N) ";R$
2 PRINT:IFR$="N"THEN6
3 OPEN4,4:PRINT#4
4 PRINT#4,"***STAMPA IMMAGINE CARATTERI***"
5 PRINT#4
6 D$="":PRINT"QUALE SET VUOI USARE: "
7 PRINT"  1  PER SET MAIUSCOLO/GRAFICO"
8 PRINT"  2  PER SET MAIUSCOLO/MINUSCOLO"
9 INPUT"■■ OPZIONE:";D$:IFD$=""THEN100
10 IFD$<>"1"ANDD$<>"2"THENPRINT"TTTT":GOTO9
14 M$=" SET MAIUSCOLO/MINUSCOLO"
15 IFD$="1"THENM$=" SET MAIUSCOLO/GRAFICO"
16 PRINT"■■SCRIVI IL CODICE DEL CARATTERE"
17 PRINT"IN D/CODE";:SW=0:INPUTD:PRINT
18 IFD<0ORD>255THENPRINT"TTTTTT":GOTO14
19 D1=D:IFD>127THEND=D-128:SW=1
21 A=14080+(VAL(D$)-1)*1024
22 FORK=0TO7:D(K)=PEEK(A+D*8+K):NEXTK
```

```
24 PRINT"[illegible]D/CODE=";D;M$:PRINT
25 PRINT"CARATTERE CORRISP. AGLI 8 BYTES:"
26 PRINT
27 PRINT"CARATTERE  VAL. BINARIO  VAL. DEC."
28 PRINT:PRINT:IFR$="N"THEN34
29 PRINT#4,"D/CODE=";D1;M$:PRINT#4
30 PRINT#4,"CARATTERE CORRISP. AGLI 8 BYTES:"
31 PRINT#4
32 PRINT#4,"  CARATTERE    VAL. BINARIO   ";
33 PRINT#4,"VAL. DEC.":PRINT#4
34 FORK=0TO7:D$(K)="":E$(K)="":B=D(K)
35 FORJ=0TO7:IFINT(B/2↑(7-J))=0THEN39
36 IFSW=1THEND$(K)=D$(K)+" ":E$(K)=E$(K)+"0":GOTO38
37 D$(K)=D$(K)+"*":E$(K)=E$(K)+"1"
38 B=B-2↑(7-J):GOTO41
39 IFSW=1THEND$(K)=D$(K)+"*":E$(K)=E$(K)+"1":GOTO41
40 D$(K)=D$(K)+" ":E$(K)=E$(K)+"0"
41 NEXTJ:NEXTK:FORK=0TO7
42 PRINTD$(K);"      ";E$(K);"       ";D(K)
43 IFR$="N"THEN50
44 PRINT#4," ";D$(K);"          ";
45 PRINT#4,E$(K);"      ";D(K)
50 NEXTK
100 IFR$="S"THENPRINT#4:CLOSE4
110 STOP
```

7.4 CONTATORI

Ci siamo proposti di estrarre un gruppo di numeri a caso, compresi nell'intervallo 0-14000, e di calcolare le frequenze con le quali i numeri cadono in uno dei 14 intervalli che seguono:

0- 1000	1001- 2000
2001- 3000	3001- 4000
4001- 5000	5001- 6000
6001- 7000	7001- 8000
8001- 9000	9001-10000
10001-11000	11001-12000
12001-13000	13001-14000

Il programma all'inizio chiede quanti numeri a caso vuoi estrarre e quale argomento iniziale vuoi usare per la funzione RND. Infatti dall'argomento iniziale dipende la sequenza:

.l'argomento negativo fa partire la sequenza di estrazione da un punto determinato; se usi lo stesso numero negativo ottiene sempre gli stessi numeri ogni volta che fai girare il programma;

.l'argomento nullo fa prelevare il seme per definire l'inizio delle estrazioni dal contatore TI, quindi questo dipende dal tempo trascorso dall'accensione

del calcolatore;
.l'argomento positivo fa proseguire l'estrazione dalla sequenza gia' in corso.
Il programma usa il seme da te fornito per eseguire un'estrazione iniziale fuori ciclo, poi prosegue con argomento positivo. Se dai un seme positivo e usi il programma subito dopo l'accensione ottieni sempre gli stessi risultati, infatti il seme iniziale e' sempre uguale. I numeri a caso estratti sono compresi tra 0 e 1; viene eseguito un semplice calcolo per ridurli nell'intervallo 0-14000. Abbiamo scelto 14000 come limite, e 14 intervalli di 1000 valori, per poter visualizzare in un solo quadro video i risultati.
Il programma conta in un contatore generale i numeri estratti e calcola in tempo reale le frequenze andando ad aggiornare i 14 contatori.
Il quadro video presenta in 14 caselle i limiti dell'intervallo, evidenziandoli in campo inverso, e sotto ad ogni casella il valore della relativa frequenza.
Puo' presentare un certo interesse vedere come andiamo a scrivere in posizioni predeterminate del video servendoci dell'istruzione CHAR in modo testo. Nel programma abbiamo introdotto in linee DATA le "coordinate" di riga e colonna dove vogliamo scrivere; le memorizziamo in due vettori e usiamo la coppia di coordinate che ci vuole per ogni contatore nell'istruzione CHAR. Ti facciamo notare che quando usi l'istruzione CHAR in modo testo non risulta valido il parametro "modo" (l'ultimo) che, invece, in modo grafico e' attivo e serve per ottenere il campo diretto o inverso. Noi per poter stampare in campo inverso dobbiamo usare i relativi caratteri di controllo nella stringa di stampa.
Per poterti spostare sul video puoi ricorrere alla CHAR, oppure usare i caratteri di controllo del cursore e le funzioni TAB e SPC.

```
1 REM CONTATORI
3 S$="                    "
5 I$="▚":O$="▬"
7 REM DESCRIZIONE CONTATORI
9 DATA "     0-- 1000"," 1001-- 2000"
11 DATA " 2001-- 3000"," 3001-- 4000"
13 DATA " 4001-- 5000"," 5001-- 6000"
15 DATA " 6001-- 7000"," 7001-- 8000"
17 DATA " 8001-- 9000"," 9001--10000"
19 DATA "10001--11000","11001--12000"
21 DATA "12001--13000","13001--14000"
```

```
23 REM LIMITI RIPARTIZIONE
25 DATA 13001,12001,11001,10001,9001,8001,7001
27 DATA 6001,5001,4001,3001,2001,1001
29 REM POSIZIONI STAMPA
31 DATA 2,1,2,21,5,1,5,21,8,1,8,21,11,1,11,21
33 DATA 14,1,14,21,17,1,17,21,20,1,20,21
35 DATA 3,1,3,21,6,1,6,21,9,1,9,21,12,1,12,21
37 DATA 15,1,15,21,18,1,18,21,21,1,21,21
39 REM CARICAMENTO MATRICI
41 DIMX(14),Y(14),Z(14),W(14)
43 RESTORE31
45 FORK=1TO14:READX(K),Y(K):NEXTK
47 RESTORE35
49 FORK=1TO14:READZ(K),W(K):NEXTK
51 REM INIZIALIZZAZIONE CONTATORI
53 DIMC(14):C=0
55 FORK=1TO14:C(K)=0:NEXTK
57 PRINT"■■■■■SCRIVI QUANTI NUMERI A CASO VUOI"
59 PRINT"ESTRARRE TRA 0 E 14000"
61 INPUT"■QUANTI: ";R
63 PRINT"■■ESTRAZIONE DI ";R;" NUMERI A CASO"
65 INPUT"■SEME PER RND: ";S
67 PRINT"■SEME PER RND: ";S
69 PRINT"■PREMI UN TASTO PER CONTINUARE"
71 GETKEYA$
73 REM PREPARAZIONE QUADRO
75 RESTORE9:PRINT"■"
77 FORK=1TO14:READA$
79 X=X(K):Y=Y(K):Z=Z(K):W=W(K)
81 CHAR,Y,X,I$+A$+D$
83 C$=S$+STR$(C(K))
85 CHAR,W,Z,RIGHT$(C$,12)
87 NEXTK
89 CHAR,2,24,"CONTATORE GENERALE: "
91 REM ESTRAZIONE NUMERO
93 N=RND(S)
95 FORJ=1TOR:N=RND(1)
97 N=1+INT(N*14000)
99 REM CERCA INTERVALLO
101 RESTORE25
103 FORK=14TO2STEP-1
105 READA
107 IFN>=ATHENC(K)=C(K)+1:X=Z(K):Y=W(K):GOTO117
109 NEXTK
111 C(1)=C(1)+1:X=Z(1):Y=W(1):K=1:GOTO117
113 NEXTJ
115 GOTO115
117 REM STAMPA CONTATORE
119 C$=S$+STR$(C(K))
121 C=C+1
123 CHAR,Y,X,RIGHT$(C$,12)
125 C$=S$+STR$(C)
127 CHAR,21,24,RIGHT$(C$,12)
129 GOTO113
```

COMMENTO A CONTATORI

.3/5: definizione S$ con 12 spazi, I$ con il carattere di controllo RVS ON, D$ con il carattere di controllo RVS OFF.

.7/21: linee DATA con le descrizioni dei 14 contatori.
.23/27: linee DATA con i limiti per il controllo delle frequenze; tali limiti sono solo 13 e in ordine decrescente, infatti i controlli si fanno in cascata partendo dal valore piu' alto.
.29/37: linee DATA con i valori delle coordinate per le posizioni video delle descrizioni dei contatori e dei contatori.
.39/49: caricamento dei dati nelle relative matrici.
.51/55: inizializzazione contatori delle frequenze.
.57/71: richiesta dati iniziali.
.73/89: preparazione quadro video iniziale.
.91/113: ciclo di estrazione dei numeri e aggiornamento contatori.
.115: ferma il programma in modo che anche premendo un tasto non si sciupa il quadro; per uscire premi STOP.
.117/129: stampa del contatore che e' stato aggiornato.

7.5 GRAFICO TRIDIMENSIONALE

Il programma 3DP, nonostante sia scritto completamente in BASIC e sia breve, e' un programma abbastanza complesso e da' risultati molto spettacolari; esso infatti disegna funzioni reali di due variabili reali, disegna cioe' funzioni tridimensionali.
Il programma disegna funzioni definite sulla parte di piano -2<X<2, -2<Y<2 che ammettano valori tra -1 e 1. Se vuoi vedere l'andamento di una funzione su un dominio piu' ampio, ti conviene modificare la linea 500, in cui e' definita la funzione. Ad esempio, se vuoi vedere la funzione f(X,Y) su un dominio -10<X<10, -10<Y<30, ti bastera' impostare la funzione f(X*5,Y*10+10)/M dove M e' il massimo valore che ABS(f(X*5,Y*10+10)) assume nell'intervallo -2<X<2, -2<Y<2 o (che e' la stessa cosa) che la funzione f(X,Y) assume nell'intervallo -10<X<10, -10<Y<30. Il programma disegna la funzione partendo dall'osservatore e procedendo verso "l'interno del monitor"; per sapere se il punto che ha disegnato e' nascosto dal disegno precedentemente fatto, controlla se il

punto che deve disegnare sullo schermo non e' compreso tra il punto piu' alto e il piu' basso di una stessa colonna di punti. Per ottenere questo avremmo bisogno di due vettori di 320 elementi ciascuno (1 per colonna di punti) in cui potremmo mettere rispettivamente il punto piu' alto e il punto piu' basso disegnato su una colonna. Poiche' la memoria non espansa del COMMODORE 16 non permette di dimensionare tali vettori (che occuperebbero 3214 byte) ci siamo riservati 640 byte di memoria in cui poniamo direttamente i valori che ci interessano (tali valori vanno da 0 a 199 quindi possono essere contenuti in un byte). Prima di presentare il listato del programma diamo ancora un'informazione che puo' aiutare a comprenderlo: il COMMODORE 16 calcola il valore della funzione tridimensionale dato un punto del piano: tale valore viene posto nella variabile Z3 e le coordinate del punto sono X3 e Y3. X3, Y3 e Z3 sono le coordinate di

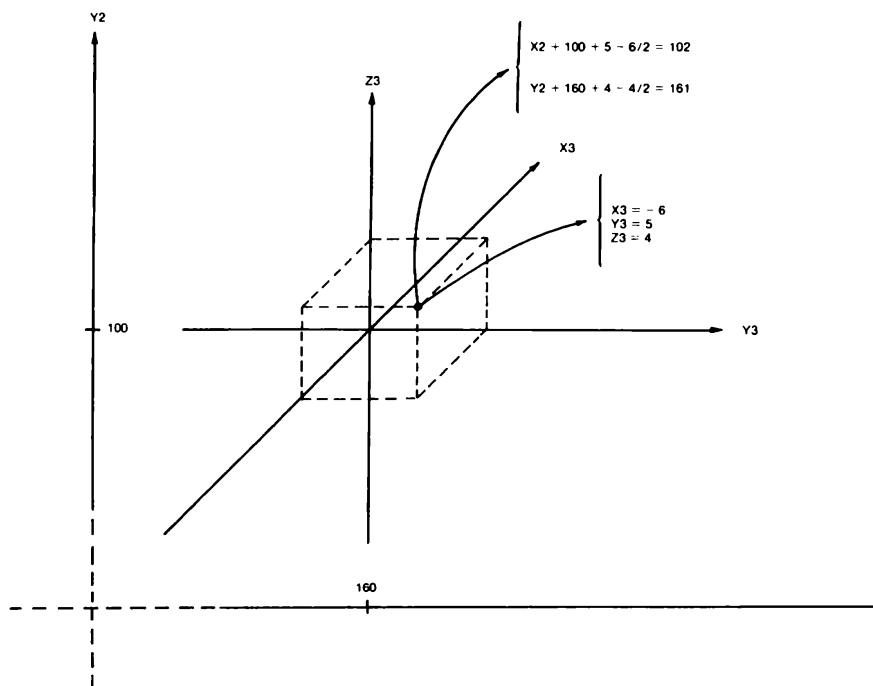



Figura 7.1 Rappresentazione in assonometria

un punto nello spazio. Il calcolatore dovra' ora disegnare un punto su un piano (il video) in modo che dia l'impressione della profondita'. Per far cio' usiamo l'assonometria, calcoliamo cioe' la coordinata verticale del piano (Y2) come Z3+X3/2 e la coordinata orizzontale del piano come (X2) come Y2+X3/2 (vedi Figura 7.1).

```
0 REM 3DP
10 POKE56,21:POKE55,128:CLR:TRAP180
20 FORI=0TO319:POKE5504+I,0:POKE5824+I,199:NEXT
30 COLOR0,1:COLOR1,2,7:COLOR4,1
40 GRAPHIC1,1
50 X3=-2:FORY3=-2TO2STEP.02:GOSUB500:NEXT
60 SP=.3
70 FORX3=-2TO2STEP.02:Y3=-2
80 GOSUB500:Y3=2:GOSUB500
90 RX=X3-INT(X3/SP)*SP
100 FORY3=-2+RXTO2STEPSP
110 GOSUB500
120 NEXTY3
130 FORY3=2-RXTO-2STEP-SP
140 GOSUB500
150 NEXTY3:NEXTX3
160 X3=2:FORY3=-2TO2STEP.02:GOSUB500:NEXT
170 GETKEYA$
180 GRAPHIC0:END
500 Z3=SIN(X3*Y3)
510 X2=160+(Y3+X3/2)*49:Y2=100+(Z3+X3/2)*49
520 IFY2>PEEK(5504+X2)THENPOKE5504+X2,Y2:GOTO550
530 IFY2<PEEK(5824+X2)THENPOKE5824+X2,Y2:GOTO560
540 RETURN
550 IFY2<PEEK(5824+X2)THENPOKE5824+X2,Y2
560 DRAW,X2,199-Y2:RETURN
```

COMMENTO A 3DP

.10: abbassa il top della memoria per creare lo spazio ai 640 byte che ci servono.

.20: pone a 0 il vettore dei massimi e a 199 quello dei minimi.

.30: schermo e bordo neri, scritte bianche.

.40: entra in modo grafico.

.50: calcola e disegna in assonometria (appoggiandosi alla routine in 500) il valore della funzione sul segmento X=-2, -2<Y<2, cioe' il segmento piu' vicino all'osservatore.

.60: pone il passo del reticolo a 0,3. Aumentando il valore della variabile SP si otterra' un reticolo a maglie piu' larghe, diminuendolo piu' strette.

.70/80: inizializza un ciclo che incrementa la X (l'asse che va dall'osservatore verso lo schermo). Quindi calcola il valore della funzione per Y=-2 e Y=2, cioe' ai lati del dominio.

.90/100: inizializza un ciclo che incrementa la Y con passo SP, partendo da un valore tale per cui il reticolo sara' diagonale rispetto agli assi.

.110: calcola e disegna la funzione.

.120: chiude il ciclo della Y.

.130/140: esegue lo stesso lavoro per i valori della Y simmetrici, per disegnare le diagonali perpendicolari alle prime.

.150: chiude i cicli della Y e della X.

.160: come la linea 50 solo che il segmento e' X=2, -2<Y<2, cioe' il segmento piu' lontano dall'osservatore.

.170: attende la pressione di un tasto.

.180: il programma salta a questa linea anche se e' stato commesso un errore o se e' stato premuto il tasto STOP. Torna al modo testo e ferma il programma.

.500: calcola il valore della funzione tridimensionale.

.510: calcola i valori di X2 e Y2.

.520: se il valore massimo della Y su quella colonna e' minore del valore appena calcolato pone il valore massimo della colonna al valore appena calcolato e salta a 550.

.530: se il valore minimo della Y su quella colonna e' maggiore del valore appena calcolato pone il valore minimo della colonna al valore appena calcolato e salta a 560.

.540: se il valore della Y2 cade tra massimo e minimo, il punto e' nascosto dalla funzione precedentemente disegnata e quindi non deve essere disegnato.

.550: il programma passa di qui se il valore della Y e' maggiore del valore massimo su quella colonna. Supera l'IF se e' anche piu' piccolo del valore minimo, cioe' se e' il primo punto di quella colonna. In tal caso pone anche il minimo uguale al valore attuale (il massimo era gia' stato sistemato nella linea 520).

.560: disegna il punto e torna.

Segue nella Figura 7.2 il risultato del programma 3DP, ottenuto con il programma VIDEOGRAF.

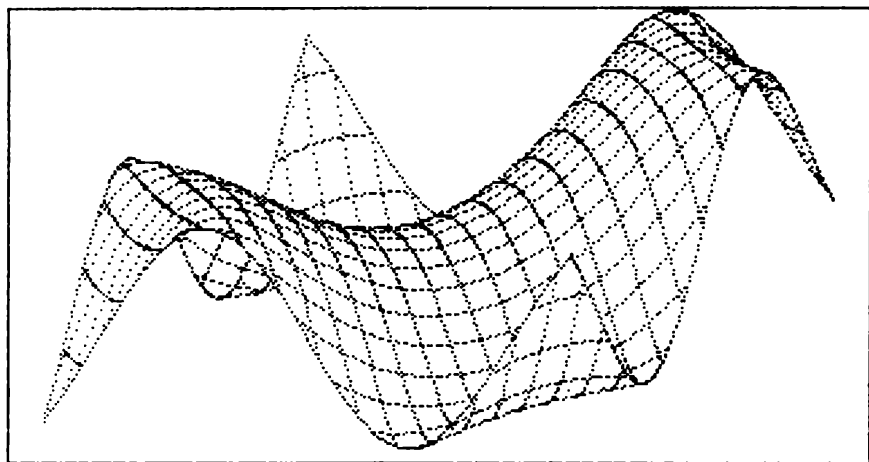



Figura 7.2 Grafico della funzione Z3=SIN(X3*Y3)

Prova a sostituire alla linea 500 le seguenti linee:

```
Z3=SIN(Y3*2)
Z3=SIN(X3*X3+Y3*Y3)
Z3=(X3*X3+Y3*Y3)*(X3*X3+Y3*Y3)/32-1
```

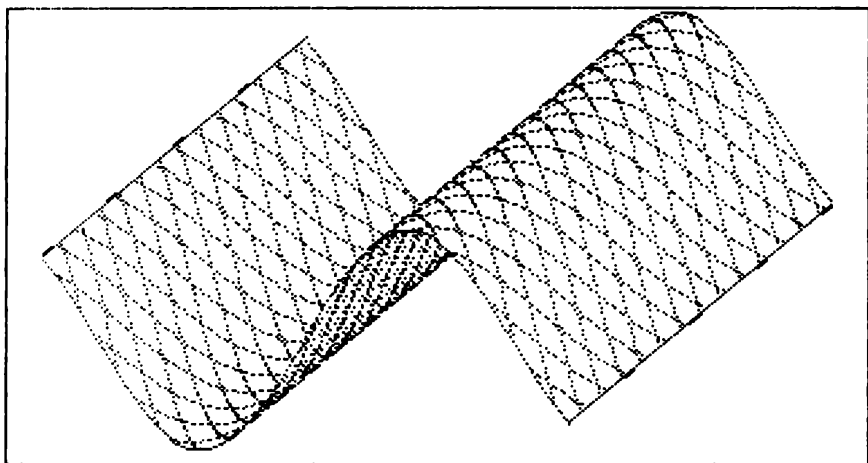



Figura 7.3 Grafico della funzione Z3=SIN(Y3*2)

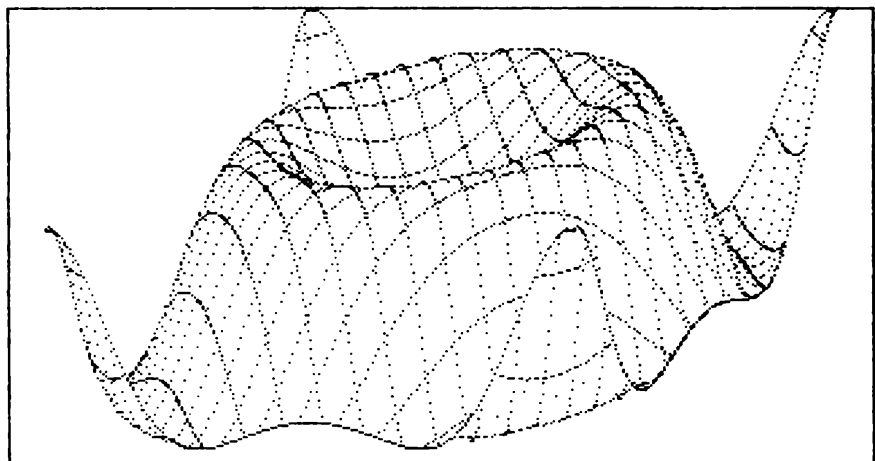



Figura 7.4 Grafico della funzione Z3=SIN(X3*X3+Y3*Y3)

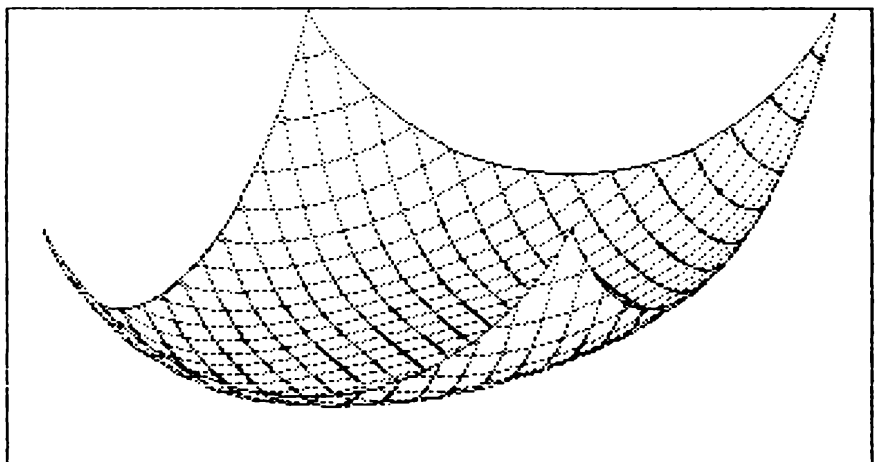



Figura 7.5 Grafico della funzione
Z3=(X3*X3+Y3*Y3)*(X3*X3+Y3*Y3)/32-1

Nelle Figure 7.3, 7.4 e 7.5 sono riportati i risultati ottenuti modificando la linea 500.

7.6 UTILIZZO INTERRUPT

Il programma INTERRUPT mostra come sia possibile, modificando opportunamente la routine di interrupt, fare svolgere al COMMODORE 16 una funzione particolare "mentre" svolge una qualunque attivita' normale (EDITOR o esecuzione di un programma). Questo programma, infatti, fa muovere un piccolo aeroplano sulla scritta "INTERRUPT MODIFICATO" che viene posta sulla prima linea dello schermo, mentre il calcolatore e' pronto a caricare e eseguire programmi, fare

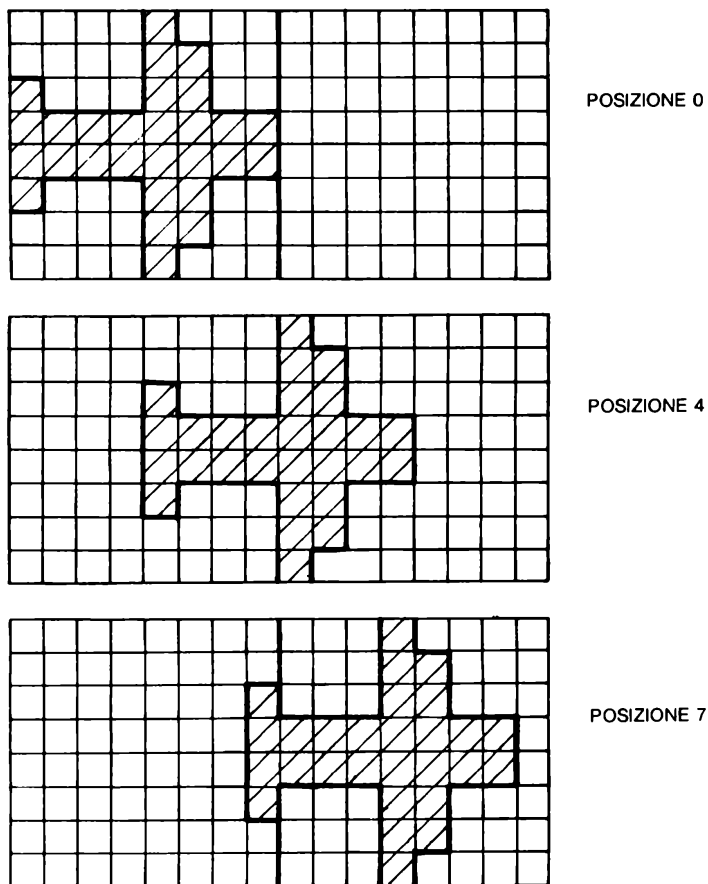



Figura 7.6 3 posizioni dell'aeroplanino

calcoli e tutto cio' che puo' fare normalmente. Per realizzare il nostro scopo "intercettiamo" la routine di interrupt, facciamo le operazioni necessarie per muovere l'aeroplano e torniamo alla normale routine di interrupt. Per intercettare la routine di interrupt basta porre nei byte $0312, $0313 l'indirizzo di partenza della routine che si vuole inserire e finire la stessa con l'istruzione JMP $CE42. Questo programma mostra anche come puoi far muovere di un punto alla volta un carattere che si sovrappone ai caratteri che incontra e che li lascia inalterati una volta che li ha superati. Il nostro aeroplanino e' lungo 8 punti: per descriverlo usiamo pero' 16 byte di memoria per poterlo far scorrere di un punto alla volta (vedi Figura 7.6).

Per ricordare quali sono i due caratteri che sono coperti dalla nostra figura usiamo 2 buffer, memorizziamo cioe' nel buffer di "destra" il carattere "sotto" la punta e nel buffer di "sinistra" quello sotto la coda; fatti otto passi avanti riportiamo la figura in posizione 0 e facciamo avanzare i due caratteri di una posizione. A questo punto dovremo mettere dove prima c'era la coda il contenuto del buffer di sinistra, nel buffer di sinistra il contenuto del buffer di destra e nel buffer di destra cio' che era davanti alla figura. Infine, per poter sovrapporre l'immagine dell'aeroplanino alle immagini dei caratteri che esso "sorvola" teniamo 16 byte in cui facciamo scorrere l'immagine dell'aeroplano. Per visualizzarlo poniamo nei byte riservati ai caratteri di D/CODE 126 e 127 l'immagine dell'aeroplano dopo aver eseguito la OR con i corrispondenti byte delle descrizioni dei caratteri i cui D/CODE sono contenti nel buffer.

Il programma BASIC INTERRUPT abbassa i puntatori di fine memoria, pone in memoria il programma in linguaggio macchina, pone la descrizione dei caratteri in RAM (da $3800 in poi; verranno pero' usati solo i caratteri minuscoli che iniziano in $3C00), seleziona il set minuscolo, scrive il messaggio, crea una finestra video, lancia la routine in linguaggio macchina e si autocancella.

Il programma in linguaggio macchina puo' essere diviso in due parti:

- la prima, INT1, (da $3B10 a $3B54) inizializza il programma.
- la seconda, INT2, (da $3B58 a $3BF9) e' la parte che verra' eseguita ad ogni chiamata di interrupt.

I byte che servono al programma in linguaggio macchina per lavorare sono:

$3B00-$3B0F: contengono la descrizione dell'aeroplano che cambia ad ogni chiamata di interrupt (vedi Figura 7.6).

$3B55: buffer di sinistra.

$3B56: buffer di destra.

$3B57: posizione dell'aeroplanino nella matrice.

$3FF0-$3FFF: sono i 16 byte che descrivono i caratteri di D/CODE 126 e 127: contengono la descrizione dell'aeroplanino dopo che ha subito la OR con le immagini dei caratteri il cui D/CODE e' nel buffer.

Ecco il listato del programma BASIC INTERRUPT:

```
0 rem interrupt
10 poke56,59:poke55,00:clr
11 fori=0to249:reada:poke15104+i,a:next
20 poke65299,(peek(65299)and3)+56
30 poke65298,peek(65298)and251
40 a$="                INTERRUPT MODIFICATO"
41 printchr$(147)chr$(14)chr$(8)a$
50 print:printchr$(27)"t":sys15120:new
1000 data8,12,204,255,255,204,12,8
1010 data0,0,0,0,0,0,0,0
1020 data120,169,212,133,4,169,60,133
1030 data6,160,0,132,3,132,5,177
1040 data3,145,5,200,208,249,230,4
1050 data230,6,165,4,201,216,208,239
1060 data169,0,141,87,59,169,12,133
1070 data4,169,0,133,3,173,0,12
1080 data141,85,59,173,1,12,141,86
1090 data59,169,59,141,19,3,169,88
1100 data141,18,3,88,96,0,0,0
1110 data173,87,59,201,8,240,87,238
1120 data87,59,162,7,94,0,59,126
1130 data8,59,202,16,247,160,7,162
1140 data1,189,85,59,133,5,169,0
1150 data133,6,6,5,38,6,6,5
1160 data38,6,6,5,38,6,169,60
1170 data24,101,6,133,6,177,5,202
1180 data208,9,25,8,59,153,248,63
1190 data76,161,59,25,0,59,153,240
1200 data63,232,202,16,204,136,16,199
1210 data160,0,169,126,145,3,200,169
1220 data127,145,3,76,66,206,169,0
1230 data141,87,59,162,7,189,8,59
1240 data157,0,59,169,0,157,8,59
1250 data202,16,242,173,85,59,160,0
1260 data145,3,173,86,59,141,85,59
1270 data230,3,165,3,201,39,208,16
1280 data169,0,133,3,173,85,59,141
1290 data39,12,173,0,12,141,85,59
1300 data160,1,177,3,141,86,59,76
1310 data109,59
```

Segue il listato del programma in linguaggio macchina, formato dalle 2 routine INT1 e INT2:

```
MONITOR

    PC   SR AC XR YR SP
; 0000  00 00 00 00 F8

. 3B10  78          SEI
. 3B11  A9 D4       LDA #$D4
. 3B13  85 04       STA $04
. 3B15  A9 3C       LDA #$3C
. 3B17  85 06       STA $06
. 3B19  A0 00       LDY #$00
. 3B1B  84 03       STY $03
. 3B1D  84 05       STY $05
. 3B1F  B1 03       LDA ($03),Y
. 3B21  91 05       STA ($05),Y
. 3B23  C8          INY
. 3B24  D0 F9       BNE $3B1F
. 3B26  E6 04       INC $04
. 3B28  E6 06       INC $06
. 3B2A  A5 04       LDA $04
. 3B2C  C9 D8       CMP #$D8
. 3B2E  D0 EF       BNE $3B1F
. 3B30  A9 00       LDA #$00
. 3B32  8D 57 3B    STA $3B57
. 3B35  A9 0C       LDA #$0C
. 3B37  85 04       STA $04
. 3B39  A9 00       LDA #$00
. 3B3B  85 03       STA $03
. 3B3D  AD 00 0C    LDA $0C00
. 3B40  8D 55 3B    STA $3B55
. 3B43  AD 01 0C    LDA $0C01
. 3B46  8D 56 3B    STA $3B56
. 3B49  A9 3B       LDA #$3B
. 3B4B  8D 13 03    STA $0313
. 3B4E  A9 58       LDA #$58
. 3B50  8D 12 03    STA $0312
. 3B53  58          CLI
. 3B54  60          RTS


MONITOR

    PC   SR AC XR YR SP
; 0000  00 00 00 00 F8

. 3B58  AD 57 3B    LDA $3B57
. 3B5B  C9 08       CMP #$08
. 3B5D  F0 57       BEQ $3BB6
. 3B5F  EE 57 3B    INC $3B57
. 3B62  A2 07       LDX #$07
. 3B64  5E 00 3B    LSR $3B00,X
. 3B67  7E 08 3B    ROR $3B08,X
. 3B6A  CA          DEX
. 3B6B  10 F7       BPL $3B64
. 3B6D  A0 07       LDY #$07
```

```
. 3B6F   A2 01      LDX #$01
. 3B71   BD 55 3B   LDA $3B55,X
. 3B74   85 05      STA $05
. 3B76   A9 00      LDA #$00
. 3B78   85 06      STA $06
. 3B7A   06 05      ASL $05
. 3B7C   26 06      ROL $06
. 3B7E   06 05      ASL $05
. 3B80   26 06      ROL $06
. 3B82   06 05      ASL $05
. 3B84   26 06      ROL $06
. 3B86   A9 3C      LDA #$3C
. 3B88   18         CLC
. 3B89   65 06      ADC $06
. 3B8B   85 06      STA $06
. 3B8D   B1 05      LDA ($05),Y
. 3B8F   CA         DEX
. 3B90   D0 09      BNE $3B9B
. 3B92   19 08 3B   ORA $3B08,Y
. 3B95   99 F8 3F   STA $3FF8,Y
. 3B98   4C A1 3B   JMP $3BA1
. 3B9B   19 00 3B   ORA $3B00,Y
. 3B9E   99 F0 3F   STA $3FF0,Y
. 3BA1   E8         INX
. 3BA2   CA         DEX
. 3BA3   10 CC      BPL $3B71
. 3BA5   88         DEY
. 3BA6   10 C7      BPL $3B6F
. 3BA8   A0 00      LDY #$00
. 3BAA   A9 7E      LDA #$7E
. 3BAC   91 03      STA ($03),Y
. 3BAE   C8         INY
. 3BAF   A9 7F      LDA #$7F
. 3BB1   91 03      STA ($03),Y
. 3BB3   4C 42 CE   JMP $CE42
. 3BB6   A9 00      LDA #$00
. 3BB8   8D 57 3B   STA $3B57
. 3BBB   A2 07      LDX #$07
. 3BBD   BD 08 3B   LDA $3B08,X
. 3BC0   9D 00 3B   STA $3B00,X
. 3BC3   A9 00      LDA #$00
. 3BC5   9D 08 3B   STA $3B08,X
. 3BC8   CA         DEX
. 3BC9   10 F2      BPL $3BBD
. 3BCB   AD 55 3B   LDA $3B55
. 3BCE   A0 00      LDY #$00
. 3BD0   91 03      STA ($03),Y
. 3BD2   AD 56 3B   LDA $3B56
. 3BD5   8D 55 3B   STA $3B55
. 3BD8   E6 03      INC $03
. 3BDA   A5 03      LDA $03
. 3BDC   C9 27      CMP #$27
. 3BDE   D0 10      BNE $3BF0
. 3BE0   A9 00      LDA #$00
. 3BE2   85 03      STA $03
. 3BE4   AD 55 3B   LDA $3B55
. 3BE7   8D 27 0C   STA $0C27
. 3BEA   AD 00 0C   LDA $0C00
. 3BED   8D 55 3B   STA $3B55
. 3BF0   A0 01      LDY #$01
. 3BF2   B1 03      LDA ($03),Y
. 3BF4   8D 56 3B   STA $3B56
. 3BF7   4C 6D 3B   JMP $3B6D
```

Nelle Figure 7.7 e 7.8 sono riportati i diagrammi a blocchi di INT1 e di INT2.

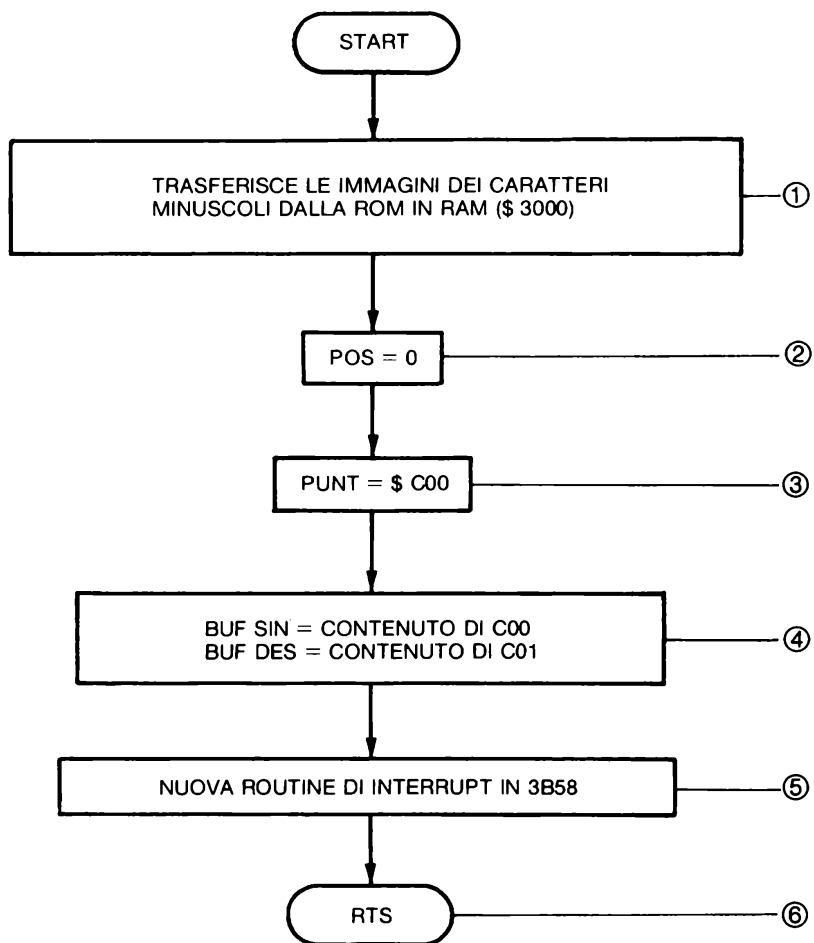



Figura 7.7 Diagramma a blocchi di INT1

Figura 7.8 Diagramma a blocchi di INT2

Ed ecco, infine, la corrispondenza tra i blocchi dei diagrammi e il listato dei programmi in linguaggio macchina:

SCHEMA INT1:
BLOCCO 1 : linee 3B10-3B2E.
BLOCCO 2 : linee 3B30-3B32.
BLOCCO 3 : linee 3B35-3B3B.
BLOCCO 4 : linee 3B3D-3B46.
BLOCCO 5 : linee 3B49-3B50.
BLOCCO 6 : linee 3B53-3B54.

SCHEMA INT2:
BLOCCO 1 : linee 3B58-3B5D.
BLOCCO 2 : linee 3BB6-3BB8.
BLOCCO 3 : linee 3BBB-3BC9.
BLOCCO 4 : linee 3BCB-3BD0.
BLOCCO 5 : linee 3BD2-3BD5.
BLOCCO 6 : linee 3BD8.
BLOCCO 7 : linee 3BDA-3BDE.
BLOCCO 8 : linee 3BE0-3BE2.
BLOCCO 9 : linee 3BE4-3BE7.
BLOCCO 10 : linee 3BEA-3BED.
BLOCCO 11 : linee 3BF0-3BF7.
BLOCCO 12 : linee 3B5F.
BLOCCO 13 : linee 3B62-3B6B.
BLOCCO 14 : linee 3B6D-3B6F.
BLOCCO 15 : linee 3B71-3B8B.
BLOCCO 16 : linee 3B8D-3BA1.
BLOCCO 17 : linee 3BA2.
BLOCCO 18 : linee 3BA3.
BLOCCO 19 : linee 3BA5.
BLOCCO 20 : linee 3BA6.
BLOCCO 21 : linee 3BA8-3BB1.
BLOCCO 22 : linee 3BB3.

APPENDICE A

ISTRUZIONI ASSEMBLER

Nelle tabelle delle pagine seguenti sono riportate le istruzioni ASSEMBLER.
Ogni riga riguarda un'istruzione, riportata in ordine alfabetico per codice mnemonico. Per ogni istruzione viene data in colonna 2 la descrizione sintetica dell'operazione. Seguono sulle colonne successive, per ogni tipo di indirizzamento, il codice esadecimale dell'istruzione, il numero dei cicli di clock impiegati per eseguirla e il numero dei byte occupati. Nelle ultime 6 colonne sono indicati i FLAG influenzati dall'esecuzione dell'istruzione.

ISTRUZIONI		immediato			assoluto			pag zero			accum			implicito			(ind,X)			(ind),Y			pag 0,X			assol,X			assol,Y			relativo			indiretto			pag 0,Y			FLAG					
mnemo	operazione	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	N	Z	C	I	D	V
ADC	A←A+M+C * ***	69	2	2	6D	4	3	65	3	2							61	6	2	71	5	2	75	4	2	7D	4	3	79	4	3										X	X	X			X
AND	A←A∧M *	29	2	2	2D	4	3	25	3	2							21	6	2	31	5	2	35	4	2	3D	4	3	39	4	3										X	X				X
ASL	C←[7 0]←0				0E	6	3	06	5	2	0A	2	1										16	6	2	1E	7	3													X	X	X			
BCC	salta se C=0**																															90	2	2												
BCS	salta se C=1**																															B0	2	2												
BEQ	salta se Z=1**																															F0	2	2												
BIT	A∧M				2C	4	3	24	3	2																															M7X					M6
BMI	salta se N=1**																															30	2	2												
BNE	salta se Z=0**																															D0	2	2												
BPL	salta se N=0**																															10	2	2												
BRK	vedi cap 5.4													00	7	1																												1		
BVC	salta se V=0**																															50	2	2												
BVS	salta se V=1**																															70	2	2												
CLC	C←0													18	2	1																											0			
CLD	D←0													D8	2	1																													0	
CLI	I←0													58	2	1																												0		
CLV	V←0													B8	2	1																														0
CMP	A-M *	C9	2	2	CD	4	3	C5	3	2							C1	6	2	D1	5	2	D5	4	2	DD	4	3	D9	4	3										X	X	X			
CPX	X-M	E0	2	2	EC	4	3	E4	3	2																															X	X	X			
CPY	Y-M	C0	2	2	CC	4	3	C4	3	2																															X	X	X			
DEC	M←M-1				CE	6	3	C6	5	2													D6	6	2	DE	7	3													X	X				
DEX	X←X-1													CA	2	1																									X	X				
DEY	Y←Y-1													88	2	1																									X	X				
EOR	A←A⊻M *	49	2	2	4D	4	3	45	3	2							41	6	2	51	5	2	55	4	2	5D	4	3	59	4	3										X	X				
INC	M←M+1				EE	6	3	E6	5	2													F6	6	2	FE	7	3													X	X				
INX	X←X+1													E8	2	1																									X	X				
INY	Y←Y+1													C8	2	1																									X	X				
JMP	salta				4C	3	3																												6C	5	3									
JSR	salta a sotprg				20	6	3																																							
LDA	A←M *	A9	2	2	AD	4	3	A5	3	2							A1	6	2	B1	5	2	B5	4	2	BD	4	3	B9	4	3										X	X				

ISTRUZIONI		immediato			assoluto			pag zero			accum			implicito			(ind,X)			(ind),Y			pag 0,X			assol,X			assol,Y			relativo			indiretto			pag 0,Y			FLAG					
mnemo	operazione	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	N	Z	C	I	D	V
LDX	X←M *	A2	2	2	AE	4	3	A6	3	2																			BE	4	3							B6	4	2	X	X				
LDY	Y←M *	A0	2	2	AC	4	3	A4	3	2													B4	4	2	BC	4	3													X	X				
LSR	0 → [7 0] → C				4E	6	3	46	5	2	4A	2	1										56	6	2	5E	7	3													0	X	X			
NOP	nessuna													EA	2	1																														
ORA	A←AVM	09	2	2	0D	4	3	05	3	2							01	6	2	11	5	2	15	4	2	1D	4	3	19	4	3										X	X				
PHA	SK←A SP←SP-1													48	3	1																														
PHP	SK←P SP←SP-1													08	3	1																														
PLA	A←SK SP←SP+1													68	4	1																									X	X				
PLP	P←SK SP←SP+1													28	4	1																									X	X	X	X	X	X
ROL	C ← [7 0] ← C				2E	6	3	26	5	2	2A	2	1										36	6	2	3E	7	3													X	X	X			
ROR	C → [7 0] → C				6E	6	3	66	5	2	6A	2	1										76	6	2	7E	7	3													X	X	X			
RTI	VEDI CAP 5.4													40	6	1																									X	X	X	X	X	X
RTS	VEDI CAP 5.4													60	6	1																														
SBC	A←A-M-C̄ *	E9	2	2	ED	4	3	E5	3	2							E1	6	2	F1	5	2	F5	4	2	FD	4	3	F9	4	3										X	X	X			X
SEC	C←1													38	2	1																											1			
SED	D←1													F8	2	1																													1	
SEI	I←1													78	2	1																												1		
STA	M←A				8D	4	3	85	3	2							81	6	2	91	6	2	95	4	2	9D	5	3	99	5	3															
STX	M←X				8E	4	3	86	3	2																												96	4	2						
STY	M←Y				8C	4	3	84	3	2													94	4	2																					
TAX	X←A													AA	2	1																									X	X				
TAY	Y←A													A8	2	1																									X	X				
TSX	X←S													BA	2	1																									X	X				
TXA	A←X													8A	2	1																									X	X				
TXS	S←X													9A	2	1																														
TYA	A←Y													98	2	1																									X	X				

* SE A CAVALLO DI PAGINA N=N+1
** SE VERIFICATO N=N+1
SE VERIFICATO E SALTA OLTRE LA PAGINA N=N+2
*** IN MODO DECIMALE IL FLAG DI ZERO NON E' VALIDO

X=REGISTRO X
Y=REGISTRO Y
A=ACCUMULATORE
M=OPERANDO
SK=CELLA DELLO STACK
SP=STACK POINTER

\+ SOMMA
\- SOTTRAZIONE
∧ AND LOGICO
V OR LOGICO
⊻ OR ESCLUSIVO

X MODIFICA
M7 BIT 7 DELLA CELLA CONSIDERATA
M6 BIT 6 DELLA CELLA CONSIDERATA
N NUMERO DI CICLI DI CLOCK
\# NUMERO DI BYTES

APPENDICE B

MAPPA DELLA MEMORIA

Label	ind. esadec.	ind. decimale	Descrizione
PDIR	$0000	0	7501 DATA DIRECTION REGISTER: direzione dei dati nella porta I/O nella CPU che si trova all'indirizzo 1. 0 = INPUT 1=OUTPUT, per ciascuno degli 8 bit
PORT	$0001	1	7501 DATA REGISTER: Registro dove leggere o scrivere lo stato dei piedini di I/O della CPU
SRCHTK	$0002	2	Temporanea per il BASIC
ZPVEC1	$0003-$0004	3-4	Temp (renumber)
ZPVEC2	$0005-$0006	5-6	Temp (renumber)
CHARAC	$0007	7	Temporanea per il BASIC
ENDCHR	$0008	8	Flag: cerca le virgolette a fine stringa
TRMPOS	$0009	9	Colonna dello schermo dall'ultimo TAB
VERCK	$000A	10	Flag: 0 = load; 1 = verify
COUNT	$000B	11	Punt. del buffer di Input/# di subscripts
DIMFLG	$000C	12	Flag: Dimensione di default per il vettore
VALTYP	$000D	13	Tipo di dato: $FF = stringa; $00 = numero
INTFLG	$000E	14	Tipo di dato: $80 = intero; $00 = floating
DORES	$000F	15	Flag: DATA scan/List quote/garbage coll
SUBFLG	$0010	16	Flag: subscript ref / user function call
INPFLG	$0011	17	Flag: $00 = INPUT; $40 = GET; $98 = READ
TANSGN	$0012	18	Flag: segno TAN / risultato del confronto
CHANNL	$0013	19	Flag: INPUT prompt
LINNUM	$0014-$0015	20-21	Temp: valore numero intero
TEMPPT	$0016	22	Puntatore: stack temporaneo per stringhe
LASTPT	$0017-$0018	23-24	Indirizzo ultima stringa temporanea
TEMPST	$0019-$0021	25-33	Stack per stringhe temporanee
INDEX1	$0022-$0023	34-35	Area per puntatori di utilita'
INDEX2	$0024-$0025	36-37	Area per puntatori di utilita'
RESHO	$0026	38	
RESMOH	$0027	39	
RESMO	$0028	40	
RESLO	$0029	41	
	$002A	42	
TXTTAB	$002B-$002C	43-44	Puntatore: inizio area programma BASIC
VARTAB	$002D-$002E	45-46	Punt: Fine programma BASIC+1 e inizio var.
ARYTAB	$002F-$0030	47-48	Punt: Fine variabili+1 e inizio vettori
STREND	$0031-$0032	49-50	Punt: Fine vettori+1
FRETOP	$0033-$0034	51-52	Punt: Fondo della zona stringhe
FRESPC	$0035-$0036	53-54	Puntatore di utilita' per le stringhe
MEMSIZ	$0037-$0038	55-56	Punt: Cima memoria riservata al BASIC+1
CURLIN	$0039-$003A	57-58	Numero della linea BASIC corrente
TXTPTR	$003B-$003C	59-60	
FNDPNT	$003D-$003E	61-62	
DATLIN	$003F-$0040	63-64	Numero della linea DATA corrente
DATPTR	$0041-$0042	65-66	Puntatore: indirizzo voce DATA corrente
INPPTR	$0043-$0044	67-68	Vettore: Routine INPUT
VARNAM	$0045-$0046	69-70	Nome della variabile BASIC corrente

VARPNT	$0047-$0048	71-72	Punt: dati della variabile BASIC corrente
FORPNT	$0049-$004A	73-74	Punt: variabile indice per FOR/NEXT
OPPTR	$004B-$004C	75-76	
OPMASK	$004D	77	
DEFPNT	$004E-$004F	78-79	
DSCPNT	$0050-$0051	80-81	
	$0052	82	
HELPER	$0053	83	
JMPER	$0054	84	
SIZE	$0055	85	
OLDOV	$0056	86	
TMPF1	$0057	87	
HIGHDS	$0058-$0059	88-89	
HIGHTR	$005A-$005B	90-91	
	$005C	92	
LOWDS	$005D-$005E	93-94	
LOWTR	$005F	95	
EXPSGN	$0060	96	
FACEXP	$0061	97	Esponente accumulatore virgola mobile 1
FACHO	$0062-$0065	98-101	Mantissa accumulatore virgola mobile 1
FACSGN	$0066	102	Segno accumulatore virgola mobile 1
SGNFLG	$0067	103	Puntatore: costante per calcolo in serie
BITS	$0068	104	Overflow accumulatore virgola mobile 1
ARGEXP	$0069	105	Esponente accumulatore virgola mobile 2
ARGHO	$006A-$006D	106-109	Mantissa accumulatore virgola mobile 2
ARGSGN	$006E	110	Segno accumulatore virgola mobile 2
ARISGN	$006F	111	Risultato confronto segni: accumul.1-acc.2
FACOV	$0070	112	Arrotondam. accumulatore virgola mobile 1
FBUFPT	$0071-$0072	113-114	Puntatore: Buffer cassetta
AUTINC	$0073-$0074	115-116	Valore di incremento per AUTO 0=escluso
MVDFLG	$0075	117	Flag: 10 K alta risoluzione allocati
KEYNUM	$0076	118	
KEYSIZ	$0077	119	
SYNTMP	$0078	120	
DSDESC	$0079-$007B	121-123	Descrittore di DS$
TOS	$007C-$007D	124-125	Cima dello stack del BASIC
TMPTON	$007E-$007F	126-127	Temporanee per la musica (tono e volume)
VOICNO	$0080	128	
RUNMOD	$0081	129	
POINT	$0082	130	
GRAPHM	$0083	131	Modo grafico corrente
COLSEL	$0084	132	Colore corrente selezionato
MC1	$0085	133	Colore multicolore 1
FG	$0086	134	Colore FOREGROUND
SCXMAX	$0087	135	Numero massimo di colonne
SCYMAX	$0088	136	Numero massimo di righe
LTFLAG	$0089	137	Flag: dipingi a sinistra
RTFLAG	$008A	138	Flag: dipingi a destra
STOPNB	$008B	139	Smette di dipingere se non sfondo
GRAPNT	$008C-$008D	140-141	
VTEMP1	$008E	142	
VTEMP2	$008F	143	
STATUS	$0090	144	Parola di stato del KERNAL (ST)
STKEY	$0091	145	Flag: Tasto STOP/tasto Reverse
SPVERR	$0092	146	Temporanea
VERFCK	$0093	147	Flag: 0=load; 1=verify
C3PO	$0094	148	Flag: Serial bus-Carattere in uscita
BSOUR	$0095	149	Carattere bufferizzato per serial bus
XSAV	$0096	150	Temporaneo
LDTND	$0097	151	Numero files aperti/indice a tabella file
DFLTN	$0098	152	Unita' di input di default (0)
DFLTO	$0099	153	Unita' di output di default (3)
MSGFLG	$009A	154	Flag: $80 = modo diretto; $00=programma
SAL	$009B	155	Errore passo 1 su nastro
SAH	$009C	156	Errore passo 2 su nastro
EAL	$009D	157	
EAH	$009E	158	
T1	$009F-$00A0	159-160	Area dati temporanea
T2	$00A1-$00A2	161-162	Area dati temporanea

TIME	$00A3-$00A5	163-165	Orologio in tempo reale in 1/60 sec.
R2D2	$00A6	166	Usata dal Serial Bus
TPBYTE	$00A7	167	Byte da essere scritto/letto su/da nastro
BSOUR1	$00A8	168	Temporanea usata dalla routine seriale
FPVERR	$00A9	169	
DCOUNT	$00AA	170	
FNLEN	$00AB	171	Lunghezza del nome del file corrente
LA	$00AC	172	Numero del file logico corrente
SA	$00AD	173	Indirizzo secondario corrente
FA	$00AE	174	Numero di unita' corrente
FNADR	$00AF-$00B0	175-176	Punt.: nome del file corrente
ERRSUM	$00B1	177	
STAL	$00B2	178	Indirizzo di inizio I/O
STAH	$00B3	179	
MEMUSS	$00B4-$00B5	180-181	Inizio della RAM da caricare
TAPEBS	$00B6-$00B7	182-183	
TMP2	$00B8-$00B9	184-185	
WRBASE	$00BA-$00BB	186-187	Puntatore ai dati per scrittura su nastro
IMPAR	$00BC-$00BD	188-189	
FETPTR	$00BE-$00BF	190-191	Punt: byte da indirizzare nell'indirizzamento a banchi
SEDSAL	$00C0-$00C1	192-193	Temporaneo per lo Scrolling
RVS	$00C2	194	Flag di reverse on
INDX	$00C3	195	
LSXP	$00C4	196	Posizione X all'inizio
LSTP	$00C5	197	
SFDX	$00C6	198	Tasto premuto
CRSW	$00C7	199	Flag: INPUT o GET dalla tastiera
PNT	$00C8-$00C9	200-201	Punt: indirizzo linea corrente di schermo
PNTR	$00CA	202	Colonna del cursore nella linea corrente
QTSW	$00CB	203	Flag: Editor in modo virgolette (0=NO)
SEDT1	$00CC	204	Uso temporaneo per editor
TBLX	$00CD	205	Numero di linea fisica del cursore
DATAX	$00CE	206	Area dati temporanea
INSRT	$00CF	207	Flag: modo Insert, >0=numero di insert
FREE	$00D0-$00E8	208-232	Area libera per l'utente
CIRSEG	$00E9	233	Tabella di collegamento linee schermo
USER	$00EA-$00EB	234-235	
KEYTAB	$00EC-$00ED	236-237	Vettore tabella scansione tastiera
TMPKEY	$00EE	238	
NDX	$00EF	239	Indice alla coda della tastiera
STPFLG	$00F0	240	Flag: pausa
TO	$00F1-$00F2	241-242	Locazione in pagina 0 usata dal MONITOR
CHRPTR	$00F3	243	
BUFEND	$00F4	244	
CHKSUM	$00F5	245	Temporanea per il controllo della parita'
LENGHT	$00F6	246	
PASS	$00F7	247	
TYPE	$00F8	248	Tipo di blocco
USEKDY	$00F9	249	
XSTOP	$00FA	250	Salva registro X per controllo rapido STOP
CURBNK	$00FB	251	Configurazione del banco corrente
XON	$00FC	252	Non usata nel COMMODORE 16
XOFF	$00FD	253	Non usata nel COMMODORE 16
SEDT2	$00FE	254	Usata temporaneamente dall'editor
LOFBUF	$00FF	255	
SYSSTK	$0100-$01FF	256-511	Stack della CPU
BUF	$0200-$0258	512-600	Buffer di input BASIC e MONITOR
	$0259-$02AC	601-684	Area usata dal BASIC per comandi DOS
	$02AD-$02F1	685-753	Area usata dal BASIC per comandi grafici
ADRAY1	$02F2-$02F3	754-755	Punt: conversione FLOATING-INTERO
ADRAY2	$02F4-$02F5	756-757	Punt: conversione INTERO-FLOATING
BNKVEC	$02FE-$02FF	766-767	Vettore per l'uso di funzioni su cartridge
IERROR	$0300-$0301	768-769	Vettore errore (codice in reg. X)
IMAIN	$0302-$0303	770-771	Vettore esecuzione istruzione BASIC
ICRNCH	$0303-$0304	772-773	Vett. tokenizzaz. (compatta parole) chiave del BASIC)
IQPLOP	$0306-$0307	774-775	Vettore routine LIST
IGONE	$0308-$0309	776-777	Vettore routine analisi prossimo carattere del programma BASIC

IEVAL	$030A-$030B	778-779	Vettore per la valutazione dei simboli
IESCLK	$030C-$030D	780-781	
IESCPR	$030E-$030F		
IESCEX	$0310-$0311		
ITIME	$0312-$0313		
CINV	$0314-$0315	788-789	Vettore di INTERRUPT
CBINV	$0316-$0317	790-791	Vettore BRK
IOPEN	$0318-$0319		VETTORI DEL SISTEMA OPERATIVO
ICLOSE	$031A-$031B		
ICHKIN	$031C-$031D		
ICKOUT	$031E-$031F		
ICLRCH	$0320-$0321		
IBASIN	$0322-$0323		
IBSOUT	$0324-$0325		
ISTOP	$0326-$0327		
IGETIN	$0328-$0329		
ICLALL	$032A-$032B		
USRCMD	$032C-$032D		
ILOAD	$032E-$032F		
ISAVE	$0330-$0331		
TAPBUF	$0333-$03F2	819-1010	Buffer per il registratore a cassetta
WRLEN	$03F3-$03F4	1011-1012	Lunghezza dei dati da scrivere su nastro
RDCNT	$03F5-$03F6	1013-1014	Lunghezza dei dati da leggere da nastro
	$03F7-$0472	1015-1138	Non usate (rs 232)
CHRGET	$0473-$0478	1139-1144	Routine di lettura caratteri da memoria usata dal BASIC per ricevere un carattere alla volta
CHRGOT	$0479-$0484	1145-1156	Seguito di CHRGET, ma entrando da questo punto ritorna l'ultimo carattere letto
USRPOK	$0500-$0503	1280-1282	Vettore di salto funzione USR
LAT	$0509-$0512	1189-1198	Tabella dei numeri di file logico
FAT	$0513-$051C	1299-1308	Tabella indirizzi primari
SAT	$051D-$0526	1309-1318	Tabella indirizzi secondari
KEYD	$0527-$0530	1319-1328	Buffer della tastiera
SAREG	$07F2	2034	Registri per il comando SYS: accumulatore
SXREG	$07F3	2035	Registro X
SYREG	$07F4	2036	Registro Y
SPREG	$07F5	2037	Registro dei FLAG
LSEM	$07FC	2044	Semaforo per arresto motore cassetta
TEDATR	$0800-0BFF	2048-3071	Mappa degli attributi per il video
TEDSCN	$0C00-0FFF	3072-4095	Mappa dei caratteri nel video

APPENDICE C

REGISTRI DEL TED

Segue una tabella che riporta gli indirizzi esadecimali e decimali dei registri del TED e la funzione dei bit di ogni registro.

INDIRIZZO HEX	DEC	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
$FF00	65280	TIMER 1 low							
$FF01	65281	TIMER 1 high							
$FF02	65282	TIMER 2 low							
$FF03	65283	TIMER 2 high							
$FF04	65284	TIMER 3 low							
$FF05	65285	TIMER 3 high							
$FF06	65286	-	1 = sfondo programmabile	1 = modo grafico	0 = annullamento dello schermo	0 = 24 righe 1 = 25 righe	posizione delle scritte per scorrimento verticale		
$FF07	65287	1 = disabilita reverse	0 = PAL 1 = NTSC	-	1 = multicolore	0 = 38 colonne 1 = 40 colonne	posizione delle scritte per scorrimento orizzontale		
$FF08	65288	LATCH DELLA TASTIERA							
$FF09	65289	INTERRUPT FLAG interrupt flag	timer 3	-	timer 2	timer 1	light pen	raster register	-
$FF0A	65290	ABILITAZIONE DEGLI INTERRUPT -	timer 3	-	timer 2	timer 1	light pen	raster register	vedi prossimo
$FF0B	65291	REGISTRO DI LINEA (scrittura per interrupt) bit 7-0 (il bit 8 e' il bit 0 del registro di abilitazione degli interrupt)							
$FF0C	65292	-	-	-	-	-	-	bit 9 della posizione del cursore	bit 8 della posizione del cursore
$FF0D	65293	POSIZIONE DEL CURSORE (bit 7-0)							
$FF0E	65294	FREQUENZA DELLA VOCE 1 (bit 7-0)							
$FF0F	65295	FREQUENZA DELLA VOCE 2 (bit 7-0)							
$FF10	65296	-	-	-	-	-	-	bit 9 frequenza voce 2	bit 8 frequenza voce 2
$FF11	65297	1 : non suona	1 : seleziona rumore (voce 2)	1 : seleziona voce 2	1 : seleziona voce 1	VOLUME			
$FF12	65298	-	-	bit 15 indirizzo mappa grafica	bit 14 indirizzo mappa grafica	bit 13 indirizzo mappa grafica	0=TED legge RAM 1=TED legge ROM	bit 9 frequenza voce 1	bit 8 frequenza voce 1
$FF13	65299	bit 15 indirizzo descr. caratteri	bit 14 indirizzo descr. caratteri	bit 13 indirizzo descr. caratteri	bit 12 indirizzo descr. caratteri	bit 11 indirizzo descr. caratteri	bit 10 indirizzo descr. caratteri	selezione clock: 1=.89, 0=1.78MHz	-
$FF14	65300	bit 15 indirizzo mappe video attr	bit 14 indirizzo mappe video attr	bit 13 indirizzo mappe video attr	bit 12 indirizzo mappe video attr	bit 11 indirizzo mappe video attr	bit 10 indirizzo mappe video attr	-	-
$FF15	65301	COLORE DELLO SCHERMO							
$FF16	65302	COLORE DI SFONDO 1							
$FF17	65303	COLORE DI SFONDO 2							
$FF18	65304	COLORE DI SFONDO 3							
$FF19	65305	COLORE DEL BORDO							
$FF1A	65306	-							
$FF1B	65307	-							
$FF1C	65308	-	-	-	-	-	-	-	bit 8 registro linea (lettura)
$FF1D	65309	REGISTRO DI LINEA (lettura) bit 7-0							
$FF1E	65310	POSIZIONE ORIZZONTALE DEL PENNELLO DEL MONITOR							
$FF3E	65342	Scrivendo un qualunque dato in questo registro la CPU seleziona la ROM							
$FF3F	65343	Scrivendo un qualunque dato in questo registro la CPU seleziona la RAM							

APPENDICE D

D / CODE

CORRISPONDENZA CARATTERI - D/CODE
PER I DUE SET DISPONIBILI

MA/GRAF		MIN/MA		D/CODE	
DIR.	INV.	DIR.	INV	DIR.	INV
@		@		0	128
A		a		1	129
B		b		2	130
C		c		3	131
D		d		4	132
E		e		5	133
F		f		6	134
G		g		7	135
H		h		8	136
I		i		9	137
J		j		10	138
K		k		11	139
L		l		12	140
M		m		13	141
N		n		14	142
O		o		15	143
P		p		16	144
Q		q		17	145
R		r		18	146
S		s		19	147
T		t		20	148
U		u		21	149
V		v		22	150
W		w		23	151
X		x		24	152
Y		y		25	153
Z		z		26	154
[		[		27	155
£		£		28	156
]		]		29	157
↑		↑		30	158
←		←		31	159
				32	160
!		!		33	161
"		"		34	162
#		#		35	163
$		$		36	164
%		%		37	165

CORRISPONDENZA CARATTERI - D/CODE
PER I DUE SET DISPONIBILI

MA/GRAF DIR.	MA/GRAF INV.	MIN/MA DIR.	MIN/MA INV	D/CODE DIR.	D/CODE INV
&		&		38	166
'		'		39	167
(		(		40	168
)		)		41	169
*		*		42	170
+		+		43	171
,		,		44	172
-		-		45	173
.		.		46	174
/		/		47	175
0		0		48	176
1		1		49	177
2		2		50	178
3		3		51	179
4		4		52	180
5		5		53	181
6		6		54	182
7		7		55	183
8		8		56	184
9		9		57	185
:		:		58	186
;		;		59	187
<		<		60	188
=		=		61	189
>		>		62	190
?		?		63	191
—		—		64	192
♠		A		65	193
│		B		66	194
—		C		67	195
—		D		68	196
—		E		69	197
—		F		70	198
│		G		71	199
│		H		72	200
╮		I		73	201
╰		J		74	202
╯		K		75	203
L		L		76	204
\		M		77	205
/		N		78	206
┌		O		79	207
┐		P		80	208
●		Q		81	209
_		R		82	210
♥		S		83	211
│		T		84	212
╭		U		85	213
X		V		86	214
O		W		87	215
♣		X		88	216
│		Y		89	217
♦		Z		90	218
+		+		91	219
▒		▒		92	220

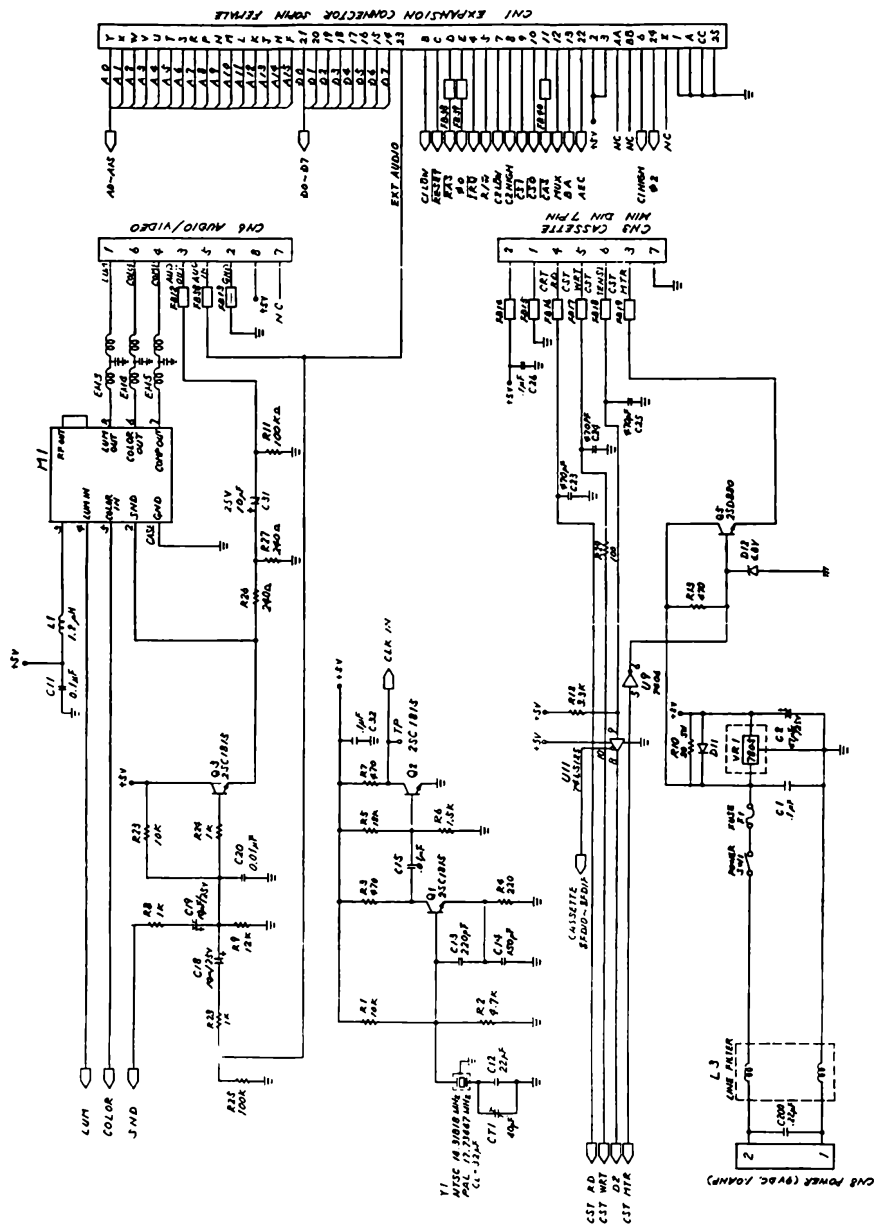
CORRISPONDENZA CARATTERI - D/CODE
PER I DUE SET DISPONIBILI

MA/GRAF DIR.	MA/GRAF INV.	MIN/MA DIR.	MIN/MA INV	D/CODE DIR.	D/CODE INV
				93	221
				94	222
				95	223
				96	224
				97	225
				98	226
				99	227
				100	228
				101	229
				102	230
				103	231
				104	232
				105	233
				106	234
				107	235
				108	236
				109	237
				110	238
				111	239
				112	240
				113	241
				114	242
				115	243
				116	244
				117	245
				118	246
				119	247
				120	248
				121	249
				122	250
				123	251
				124	252
				125	253
				126	254
				127	255

APPENDICE E

SCHEMA ELETTRICO

Segue lo schema elettrico del COMMODORE 16.

CPU 7501 U2
TED 7360 U1
U13 6529B
U7 74LS257
U8 74LS257

Il libro fa seguito al primo volume: "Commodore 16 per te", cioè presuppone una conoscenza elementare del BASIC. Per evitare ripetizioni di argomenti già trattati, si fa riferimento, dove necessario, al primo volume. In questa sede viene approfondita la conoscenza del BASIC 3.5, riportando molti programmi esempio, opportunamente commentati.
Viene affrontato in modo approfondito l'uso delle periferiche, dedicando un capitolo alla stampante e uno alla gestione dei file su disco.
Uno degli argomenti interessanti e istruttivi del libro è la programmazione in linguaggio macchina, che viene affrontato fornendo le notizie tecniche necessarie per applicarla, e parecchi esempi. Sono illustrate le istruzioni riconosciute dal microprocessore e sono presenti tutti i riferimenti necessari alla struttura hardware. Inoltre sono spiegate le funzioni delle più importanti routine del sistema operativo.
Viene descritto l'uso del processore video, cosa che permette di raggiungere risultati grafici altrimenti non ottenibili in BASIC.

L. 35.000

Cod. 427B ISBN 88-7056-278-6